МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

Савченко А.С
« » _____2020 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО СТУПЕНЯ «МАГІСТР»

Тема: **«Web - додаток car-dealer»**

Виконавець: Цуран Андрій Вікторович

Керівник: к.т.н., доцент Воронін Альберт Миколайович

Нормоконтролер : Райчев І.Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій
Спеціальність 122 "Комп'ютерні науки та інформаційні технології"
Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

Завідувач випускової кафедри
Савченко А.С.
« » 2020 р.

ЗАВДАННЯ

на виконання дипломного проекту студента

Цуран Андрія Вікторовича

(прізвище, ім'я, по батькові)

1. Тема проекту: «Web - додаток car-dealer.» затверджена наказом ректора №2175/ст. від 14.10.2019р.

2. Термін виконання роботи: з 14.10.2019р. по 09.02.2020р.

3. Вихідні дані роботи: розроблений web-додаток Car-dealer

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): Обґрунтування розробки розробки web-додатоку. Аналіз технологій та предметної області web-додатоку. Розробка web-додатоку car-dealer. Висновки.

5. Перелік обов'язкового графічного матеріалу: інформативні рисунки, приклади коду, презентація в MS PowerPoint.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1.	Проаналізувати літературу та джерела за темою дипломного проекту.	13.10.2019р – 20.10.2019р.	
2.	Розроблення та затвердження плану дипломного проекту.	21.10.2019р. – 22.10.2019р.	
3.	Провести консультації з науковим керівником щодо створення першого розділу.	21.10.2019р.	
4.	Розробка розділу 1: Обґрунтування розробки розробки web-додатоку	22.10.2019р. – 25.11.2019р.	
5.	Розробка розділу 2: Аналіз технологій та предметної області web-додатоку	26.11.2020р. – 09.01.2020р.	
6.	Розробка розділу 3: Розробка web-додатоку car-dealer	09.01.2020р. – 15.01.2020р.	
7.	Висновки та оформлення пояснювальної записки дипломного проекту.	15.01.2020р. – 27.01.2020р.	
8.	Підписання необхідних документів у встановленому порядку.	27.01.2020р. – 31.01.2020р.	
9.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту	31.01.2020р.	

7. Дата видачі завдання: 13.10.2019 р.

Керівник дипломного проекту _____ Воронін А.М.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Цуран А.В.

(підпис випускника) (П.І.Б.).

РЕФЕРАТ

Пояснювальна записка до дипломного проекту роботи «Web - додаток Car-dealer.» викладена на 96 сторінку, містить 35 рисинків.

Ключеві слова: WEB-ДОДАТОК, РОЗРОБКА, РЕАЛІЗАЦІЯ, ПРОЕКТУВАННЯ, ТЕХНОЛОГІЯ.

Об'єкт дослідження: технологія розробки для веб-додатку на мові Java.

Предмет дослідження: веб-додаток для продажу машин.

Мета роботи: створити веб-додаток для продажу машин.

Методи дослідження: об’єктно-орієнтований аналіз і проектування.

Отримані результати: реалізовано і введенний в експлуатацію веб-додаток, який дозволяє зручно купувати та продавати автомобілі.

Результати дипломної роботи планується використовувати: при подальшій розробці проектів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ARM	*Advanced RISC Machine*
AWT	*Abstract Window Toolkit*
CASE	*Computer-Aided Software Engineering*
DOCS	*Documents*
GUI	*Graphical User Interface*
IDE	*Integrated Development Environment*
J2EE	*Java 2 Enterprise Edition*
JDBC	*Java DataBase Connectivity*
JIT	*Just-In-Time compilation*
JVM	*Java Virtual Machine*
SDK	*Software Development Kit*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*
БД	База даних
ЕОМ	Електронна обчислювальна машина
ІС	Інформаційна система
ОЗП	Оперативно-запам’ятовуючий пристрій
СУБД	Система управління базами даних

ВСТУП

Проблема автоматизації виробничих процесів і процесів управління як засобу підвищення продуктивності праці завжди була і залишається актуальною.

Автоматизація – один з напрямів науково-технічного прогресу, спрямований на застосування саморегульованих технічних засобів, економіко- математичних методів і систем керування, що звільняють людину від участі в процесах отримання, перетворення, передачі і використання енергії, матеріалів чи інформації, істотно зменшують міру цієї участі чи трудомісткість виконуваних операцій. Разом з терміном автоматичний, використовується поняття автоматизований, що підкреслює відносно великий ступінь участі людини в процесі.

Сьогодні продуктивність є головним фактором, який визначає ефективність системи. Правильне проектне рішення служить основою високопродуктивної системи.

В реальних умовах проектування – це пошук способу забезпечити вимоги функціональності системи засобами наявних технологій із врахуванням заданих обмежень.

Метою даного дипломного проекту є розробка бекенду для Web-додатку «Task-list». Web-додаток призначений для управління проектом. В контексті управління проектом означає систему візуального управління процесом розробки проекту.

РОЗДІЛ 1

ОБҐРУНТУВАННЯ РОЗРОБКИ WEB-ДОДАТКУ

1.1 Поняття «Web-додаток»

Веб-додаток — розподілений додаток, в якому клієнтом виступає браузер, а сервером — веб-сервер. Браузер може бути реалізацією так званих тонких клієнтів — логіка додатку зосереджується на сервері, а функція браузера полягає переважно у відображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є міжплатформовими сервісами. Унаслідок цієї універсальності та відносної простоти розробки веб-застосунки стали широко популярними в кінці 1990-х — початку 2000-х років. [1]

Сьогодні веб-додатки набирають популярність. найбільш відвідуваними сайтами стають не чисто інформаційні, гіпертекстові сайти, а ті, які надають будь-які сервіси, будь-яким чином взаємодіють з користувачем. Але навіть і звичайні інформаційні сайти часто використовують системи управління контентом для зручності управління інформацією, так що і їх теж можна зарахувати до веб-додатків.

Веб-додаток являє собою веб-сайт, на якому розміщені сторінки з частково або повністю несформованим вмістом. Остаточний вміст формується тільки після того, як відвідувач сайту запросить сторінку з веб-сервера. У зв'язку з тим що остаточний вміст сторінки залежить від запиту, створеного на основі дій користувача, така сторінка називається динамічної.

Будь-який веб-додаток являє собою набір статичних і динамічних веб-сторінок. Статична веб-сторінка - це сторінка, яка завжди відображається перед користувачем в незмінному вигляді. Веб-сервер відправляє сторінку за запитом веб-браузера без будь-яких змін. На противагу цьому, сервер вносить зміни в динамічну веб-сторінку перед відправкою її браузеру. У зв'язку з тим що сторінка змінюється, вона називається динамічною.

Коли веб-сервер отримує запит на видачу статичної веб-сторінки, він відправляє сторінку безпосередньо браузеру. Однак, коли запитується динамічна сторінка, дії веб-сервера не настільки однозначні. Сервер передає сторінку спеціальною програмою, яка і формує остаточну сторінку. Така програма називається сервером додатків.

Сервер додатків виконує читання коду, що знаходиться на сторінці, формує остаточну сторінку відповідно до прочитаним кодом, а потім видаляє його з сторінки. В результаті всіх цих операцій виходить статична сторінка, яка передається веб-сервера, який в свою чергу відправляє її клієнтському браузеру.

Процес включає наступні етапи:

1. Веб-браузер запитує динамічну сторінку.

2. Веб-сервер знаходить сторінку і передає її сервера додатків.

3. Сервер додатків переглядає сторінку на наявність інструкцій і виконує її створення.

4. Сервер додатків повертає підготовлену сторінку на веб-сервер.

5. Веб-сервер відправляє підготовлену сторінку запити її браузеру.

Всі сторінки, які отримує браузер, містять тільки HTML-код. Схематичне зображення процесу на рис.1.1.

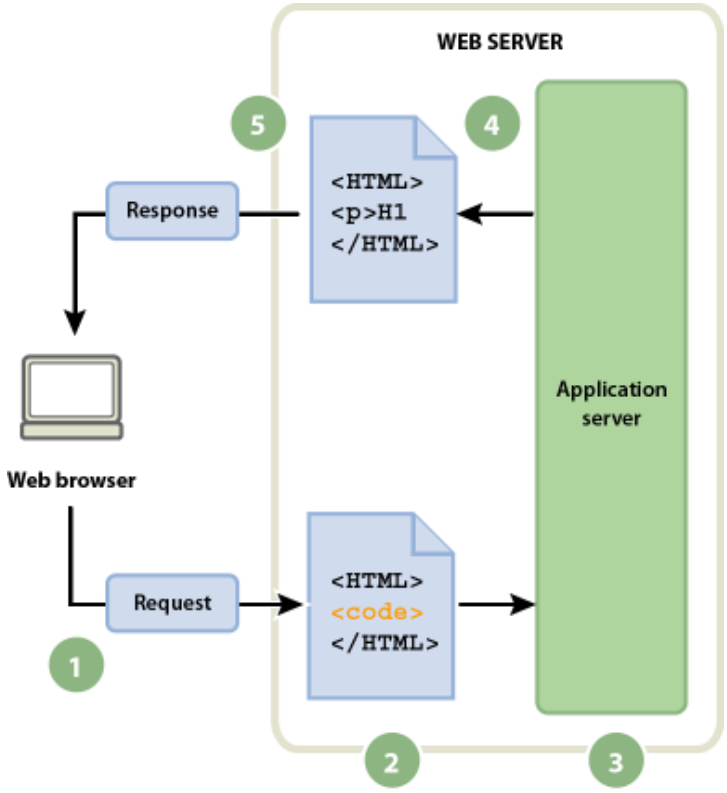

Рис.1.1 Схематичне зображення процесу

Сервер додатків надає можливість використовувати такі ресурси сервера, як бази даних. Наприклад, динамічна сторінка може містити програмні інструкції для сервера додатків, слідуючи яким сервера необхідно отримати певні дані з бази даних і помістити їх в HTML-код сторінки.

Зберігання вмісту в базі даних дозволяє відокремити оформлення веб-сайту від вмісту, яке будуть бачити користувачі. Замість того щоб створювати всі сторінки у вигляді окремих HTML-файлів, пишуться тільки шаблони сторінок для кожного виду інформації, що представляється. Потім вміст завантажується в базу даних, після чого веб-сайт буде витягувати його при запитах користувачів. Крім того, можна оновити інформацію в одному джерелі і продублювати це зміна на всіх веб-сайті без редагування кожної сторінки вручну. Adobe Dreamweaver дозволяє створювати веб-форми для вставки, оновлення та видалення інформації в базі даних.

Програмна інструкція, призначена для отримання даних з бази даних, називається запитом до бази даних. Запит складається з критеріїв пошуку, виражених за допомогою мови баз даних, званого SQL (мова структурованих

запитів). Текст SQL-запиту розташовується в сценаріях сторінок на стороні сервера або в тегах.

Сервер додатків не може безпосередньо отримати дані з бази, оскільки бази даних використовують специфічні формати зберігання даних. Тому для підключення до бази даних сервер додатків використовує посередника - драйвер бази даних. Драйвер бази даних являє собою програмний модуль, за допомогою якого встановлюється взаємодія між сервером додатків і базою даних.

Після того як драйвер встановить з'єднання, виконується запит до бази, в результаті чого формується набір записів. Набір записів є безліч даних, отриманих з однієї або декількох таблиць бази даних. Набір записів повертається сервера додатків, який використовує отримані дані для формування сторінки. [2]

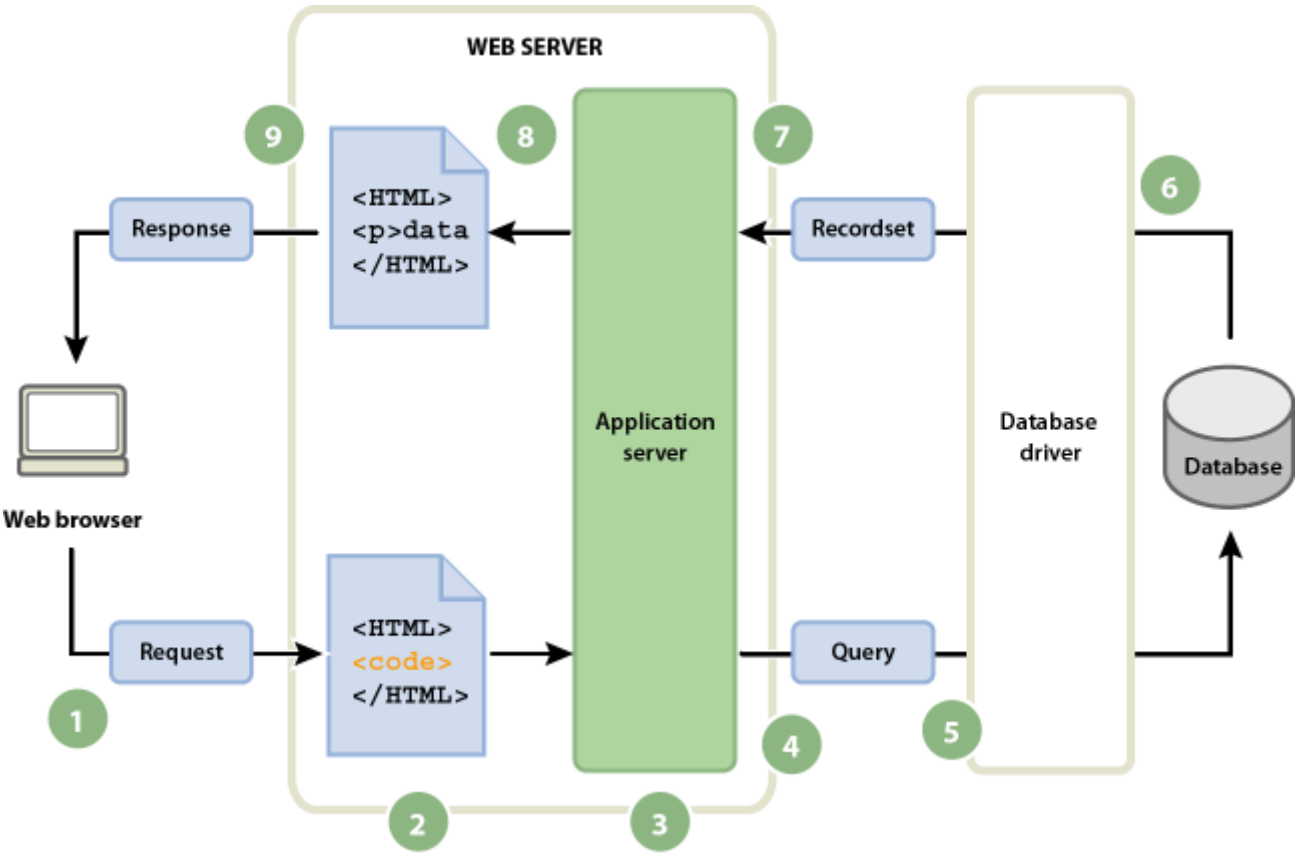

Рис.1.2 Процес формування сторінки

1. Веб-браузер запитує динамічну сторінку.

2. Веб-сервер знаходить сторінку і передає її сервера додатків.

3. Сервер додатків переглядає сторінку на наявність інструкцій і виконує її підготовку.

4. Сервер додатків відправляє запит драйверу бази даних.

5. Драйвер виконує запит в базі даних.

6. Драйвера повертається набір записів.

7. Драйвер передає набір записів сервера додатків.

8. Сервер додатків вставляє дані в сторінку і передає сторінку веб-сервера.

9. Веб-сервер відправляє підготовлену сторінку запити її браузеру.

Для використання в веб-додатку придатна будь-яка база даних за умови, що на сервері встановлений відповідний драйвер бази даних.

Для створення додатку «car-dealer» було використанно Amazon Relational Database Service.

Це розподілена реляційна база даних (РБД), яка надається як сервіс від Amazon Web Services (AWS). Є вебсервісом, який виконується у «хмарі». Спеціально розроблений для спрощення установки, простоти обслуговування та легкого масштабування РБД в застосунках. Такі складні процеси, як оновлення програмного забезпечення бази даних, проведення резервного копіювання, повернення до раннього стану (відновлення) відбувається автоматично. Масштабування дискового простору та процесорних ресурсів може бути виконано через API системи, оскільки AWS не передбачає ssh-з'єднання з екземплярами RDS

1.2 Архітектурний шаблон Model-view-Controller

Шаблон проектування MVC передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: Модель, Представлення і Контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно (рис.1.3).

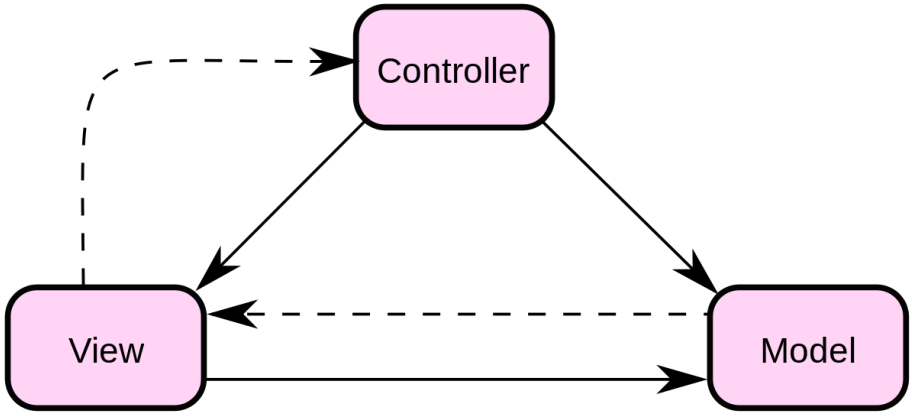

Рис.1.3 Шаблон проектування MVC

В даному шаблоні модель не залежить від представлення або керуючої логіки, що робить можливим проектування моделі як незалежного компонента і, наприклад, створювати кілька представленнь для однієї моделі.

Вперше цей шаблон був застосований в фреймворку, що розробляється для мови Smalltalk в кінці 1970-х років. З цього моменту він відіграє основну роль в більшості фреймфорков з призначеним для користувача інтерфейсом. Він докорінно змінив погляд на проектування додатків.

Модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних.

Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані.

У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля. [3]

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач

через контролер внесе зміни до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

Для реалізації шаблону було використано фреймворк Spring MVC. Він забезпечує роботу цього паттерну. Вся логіка роботи Spring MVC побудована навколо DispatcherServlet, який приймає і обробляє всі HTTP-запити (з UI) і відповіді на них. Робочий процес обробки запиту DispatcherServlet'ом проілюстрований на наступній діаграмі:

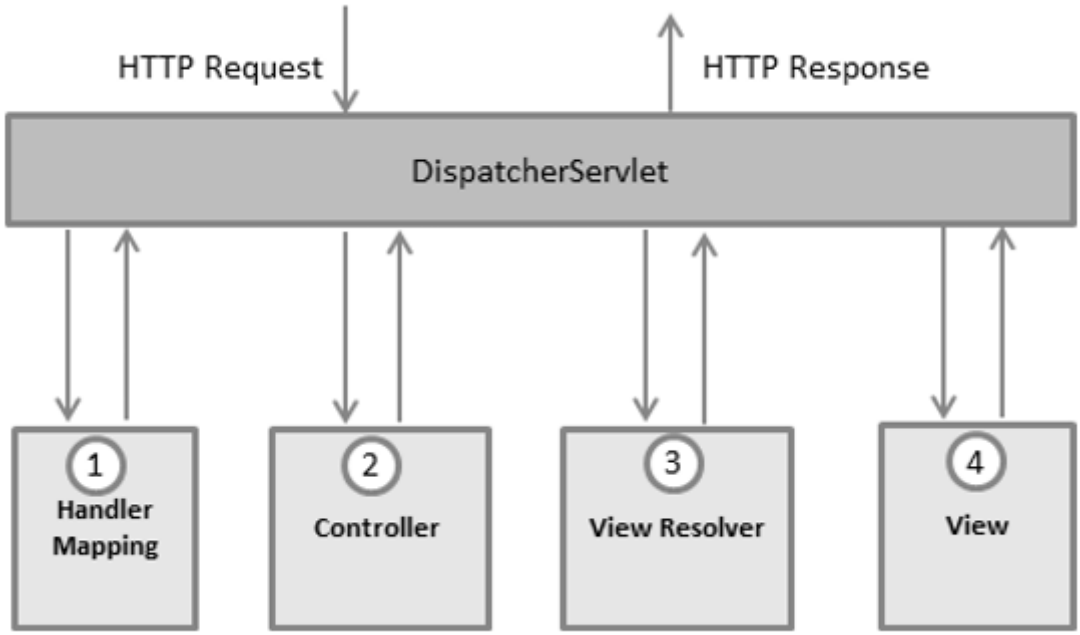

Рис.1.4 Робочий процес обробки запиту DispatcherServlet'ом

Нижче наведена послідовність подій, відповідна входить HTTP-запиту:

- Після отримання HTTP-запиту DispatcherServlet звертається до інтерфейсу HandlerMapping, який визначає, який Контролер повинен бути викликаний, після чого, відправляє запит в потрібний Контролер.
- Контролер приймає запит і викликає відповідний службовий метод, заснований на GET або POST методах. Викликаний метод визначає дані Моделі, засновані на певній бізнес-логіці і повертає в DispatcherServlet ім'я Виду (View).

- За допомогою інтерфейсу ViewResolver DispatcherServlet визначає, який Вид потрібно використовувати на підставі отриманого імені.
- Після того, як Вид (View) створений, DispatcherServlet відправляє дані Моделі у вигляді атрибутів в Вид, який в кінцевому підсумку відображається в браузері.

Всі вище згадані компоненти, а саме, HandlerMapping, Controller і ViewResolver, є частинами інтерфейсу WebApplicationContext extends ApplicationContext, з деякими додатковими особливостями, необхідними для створення web-додатків.

1.2 Мікросервісна архітектура для Web-додатку

Мікросервіси — архітектурний стиль за яким єдиний застосунок будується як сукупність невеличких сервісів кожен з яких працює у своєму власному процесі і комунікує з рештою використовуючи легковагові механізми, зазвичай HTTP. Ці сервіси будуються навколо бізнес-потреб і розгортаються незалежно з використанням зазвичай повністю автоматизованого середовища. Існує абсолютний мінімум централізованого керування цими сервісами. Самі по собі вони можуть бути написані з використанням різних мов і технологій зберігання даних.

Мікросервісна архітектура добре підходить для процесу безперервної поставки, на відміну від сервіс-орієнтовної архітектури мікросервісна спрямована на створення одного застосунка в той час як сервісно орієнтована система — являє собою множину застосунків які взаємодіють між собою.

Основні властивості:

- Високий рівень незалежності: незалежна розробка, незалежне розгортання
- Незалежне масштабування
- Невелика кодова база зменшує кількість конфліктів та дозволяє швидко

залучати нових розробників

- Простота заміни однієї реалізації сервісу іншою
- Простота додавання нового функціоналу в систему
- Ефективне використання ресурсів
- Еластичність: вихід з ладу одного сервісу зазвичай не призводить до виходу з ладу всієї системи
- Сервіси організовані відносно бізнес логіки яку вони виконують
- Кожен сервіс незалежно від інших може бути реалізований за допомогою будь-якої мови програмування, СУБД, та ін.
- Архітектурно побудовані за симетричним принципом (виробник-споживач)

Простий приклад налаштування системи мікропослуг за допомогою Spring, Spring Boot та Spring Cloud.

Мікросервіси дозволяють будувати великі системи з ряду взаємодіючих компонентів. На рівні процесу це робить те, що Spring завжди робився на рівні компонентів: нещільно зв'язані процеси замість слабко з'єднаних компонентів.

Додаток для покупок

Наприклад, уявіть інтернет-магазин з окремими мікросервісами для облікових записів користувачів, оформлення замовлень у каталозі продуктів та кошики для покупок:

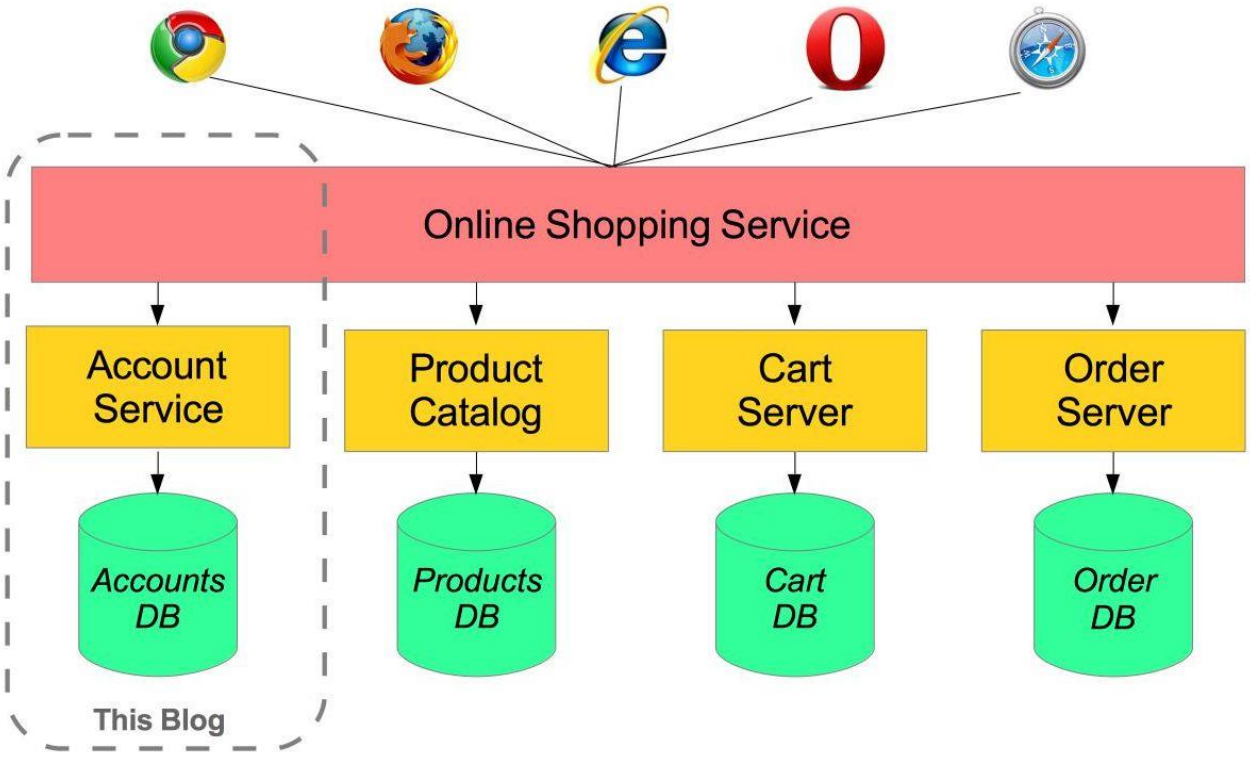

Рис.1.5 Приклад мікросервісної архітектури

Цей додаток являє собою розподілену систему, яка складається з такийх компонетів: Account Service, Product Catalog, Cart Server, Order Server та головний сервіс Online Shopping Service. Головний компонет насамперед працює з клієнтом, розподілює HTTP запити між сервісами. Acount Service відповідає за данні користувачів, реєстрацію нових юзерів. Product Catalog Service це каталог товарів для додатку. Cart Server – компонет який відповідає за оплату покупок користувачів. За допомогою сервісу Order Server можна створити та перевірити статус замовлення.

Головною перевагою такої архітектури є те що система розподілена та може працювати без деяких компонентів, це значна перевага над монолітною системою.

ВИСНОВОК ДО РОЗДІЛУ 1

Сьогодні web-додатки є актуальною темою. Користувачі на сьогодні мають потребу використовувати динамічні веб-сайти. Істотною перевагою побудови веб-застосунків для підтримки стандартних функцій браузера є те, що функції повинні виконуватися незалежно від операційної системи клієнта.

Вибрана архітектура MVC для web-додатку є досить зручною і розподіленою. Шаблон повинен полегшувати подальші зміни або розширення web-додатку, а також надавати можливість повторного використання окремих компонентів додатку.

У першому розділі був проведенний аналіз основних принципів роботи web-додатку.

РОЗДІЛ 2

АНАЛІЗ ТЕХНОЛОГІЙ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Angular

Для розуміння що таке фреймворк Angular треба розібратись в його основі, а саме TypeScript.

Рис.2.1 Логотип мови програмування TypeScript

TypeScript - мова програмування з відкритим кодом, розроблена та підтримувана Microsoft. Це суворий синтаксичний набір JavaScript і додає до мові необов'язкове статичне введення тексту.

TypeScript призначений для розробки великих програм та транскопіляцій у JavaScript. Оскільки TypeScript є набором JavaScript, існуючі програми JavaScript також є дійсними програмами TypeScript. TypeScript може використовуватися для розробки програм JavaScript для виконання як на стороні клієнта, так і на стороні сервера (як у Node.js або Deno).

Існує кілька варіантів транскопіляції. Може використовуватися перевірка TypeScript за замовчуванням, або компілятор Babel може бути викликаний для перетворення TypeScript в JavaScript.

TypeScript підтримує файли визначення, які можуть містити інформацію про типи існуючих бібліотек JavaScript, подібно до файлів

заголовків C ++, можуть описувати структуру існуючих об'єктних файлів.

Це дає можливість іншим програмам використовувати значення, визначені у файлах, так, як якщо б вони були статично набраними сутностями TypeScript.

Існують сторонні файли заголовків для популярних бібліотек, таких як jQuery, MongoDB і D3.js. Також доступні заголовки TypeScript для основних модулів Node.js, що дозволяє розробляти програми Node.js в TypeScript.

Компілятор TypeScript сам пишеться в TypeScript і компілюється в JavaScript. Він ліцензований за ліцензією Apache 2.0.

TypeScript включений як першокласна мова програмування в Microsoft Visual Studio 2013 Update 2 та новіших версій, поряд із C # та іншими мовами Microsoft. Офіційне розширення дозволяє Visual Studio 2012 також підтримувати TypeScript.

Андер Хейлсберг, провідний архітектор C # і творець Delphi і Turbo Pascal, працював над розробкою TypeScript.

Вперше TypeScript був оприлюднений у жовтні 2012 року (у версії 0.8) після двох років внутрішньої розробки в Microsoft. Незабаром після оголошення Мігель де Іказа похвалив саму мову, але розкритикував відсутність зрілої підтримки IDE, окрім Microsoft Visual Studio, яка в той час не була доступною для Linux та OS X. Сьогодні існує підтримка в інших IDE, зокрема в Eclipse, за допомогою плагіна, який надає Palantir Technologies. Різні текстові редактори, включаючи Emacs, Vim, Webstorm, Atom та власний код Visual Studio Microsoft, також підтримують TypeScript.

TypeScript 0.9, випущений у 2013 році, додав підтримку для дженериків. TypeScript 1.0 був випущений на конференції розробників Microsoft у 2014 році. Visual Studio 2013 Update 2 забезпечує вбудовану підтримку TypeScript.

У липні 2014 року команда розробників оголосила про новий компілятор TypeScript, який вимагає 5 × підвищення продуктивності. Одночасно

вихідний код, який спочатку розміщувався на CodePlex, був переміщений до GitHub.

22 вересня 2016 року було випущено TypeScript 2.0; він ввів декілька функцій, включаючи можливість програмістів необов'язково запобігати присвоєнню змінним нульових значень [26], іноді їх називають помилкою в мільярд доларів.

TypeScript виникла з недоліків JavaScript для розробки масштабних додатків як у Microsoft, так і серед їх зовнішніх клієнтів. Проблеми, що стосуються роботи зі складним кодом JavaScript, спричинили попит на користувальницькі інструменти для полегшення розробки компонентів мови.

Розробники TypeScript шукали рішення, яке не порушило б сумісність зі стандартом та його міжплатформою підтримкою. Знаючи, що поточна стандартна пропозиція ECMAScript обіцяє майбутню підтримку програмування на основі класів, TypeScript базувався на цій пропозиції. Це призвело до компілятора JavaScript з набором синтаксичних розширень мови, суперсети на основі пропозиції, що перетворює розширення в звичайний JavaScript. У цьому сенсі TypeScript був попереднім переглядом того, чого можна очікувати від ECMAScript 2015. Унікальним аспектом, який не в пропозиції, а доданий до TypeScript, є необов'язкове статичне введення тексту, що дозволяє статичний аналіз мови, що полегшує інструментарій та підтримку IDE.

TypeScript - це розширення мови, яке додає функції до ECMAScript 6. До додаткових функцій належать:

- Інтерфейси
- Перелічені типи
- Дженріки
- Простори імен
- Кортежі

- Асинхронізація / очікування.

Рис.2.2 Логотип фремворку Angular

Angular - це структура веб-додатка з відкритим кодом на основі TypeScript, яку очолює команда Angular у Google та спільнота осіб та корпорацій. Angular - це повний перезапис з тієї ж команди, яка побудувала AngularJS.

Angular був розроблений як замислений перепис AngularJS.

- Angular не має поняття "область" або контролерів, натомість він використовує ієрархію компонентів як свою первинну архітектурну характеристику.
- Angular має інший синтаксис вираження, зосереджений на "[]" для прив'язки властивостей та "()" для прив'язки подій.
- Модульність - багато основних функціональних можливостей перенесли на модулі
- Angular рекомендує використовувати мову TypeScript Microsoft, яка містить такі функції:

- Статичне введення тексту, включаючи Generics
- Анотації
- TypeScript - це супернабір ECMAScript 6 (ES6) і назад сумісний з ECMAScript 5 (тобто. JavaScript).
- Динамічне навантаження
- Асинхронні компіляції шаблонів
- Ітеративні зворотні виклики, надані RxJS. RxJS обмежує видимість стану та налагодження, але їх можна вирішити за допомогою реактивних додатків, таких як ngReact або ngrx.
- Підтримка Angular Universal, який запускає програми на серверах.

Назва

Спочатку перезапис AngularJS командою називали "Angular 2", але це призвело до плутанини серед розробників. Для уточнення, команда оголосила, що для кожної основи слід використовувати окремі терміни з "AngularJS" з посиланням на версії 1.X та "Angular" без "JS" з посиланням на версії 2 і вище.

Версія 2

Angular 2.0 був оголошений на конференції ng-Europe 22-23. Жовтень 2014. Різкі зміни у версії 2.0 викликали значну суперечку серед розробників. 30 квітня 2015 року розробники Angular оголосили, що Angular 2 перейшов від Alpha до попереднього перегляду для розробників. Angular 2 перейшов до бета-версії в грудні 2015 року, а перша версія на реліз був опублікований у травні 2016 року. Остаточна версія була опублікована 14 вересня 2016 року.

Версія 4

13 грудня 2016 року було оголошено Angular 4, пропускаючи 3, щоб уникнути плутанини через невідповідність версії пакета маршрутизатора, яка вже поширювалася як v3.3.0. Остаточна версія була опублікована 23 березня

2017 року.

Angular 4.3 - це другорядний випуск, що означає, що він не містить жодних змін, і що це замінна версія 4.x.x.

Особливості у версії 4.3

Представляємо HttpClient, меншу, простішу у використанні та більш потужну бібліотеку для подання HTTP-запитів.

Нові події життєвого циклу маршрутизатора для Guards та Resolvers. Чотири нові події: GuardsCheckStart, GuardsCheckEnd, ResolveStart, ResolveEnd приєднаються до наявного набору подій життєвого циклу, таких як NavigationStart.

Версія 5

Angular 5 вийшов 1 листопада 2017 року. Основні вдосконалення в Angular 5 включають підтримку прогресивних веб-додатків, оптимізатор збірки та вдосконалення, пов'язані з дизайном матеріалів.

Версія 6

Angular 6 вийшов 4 травня 2018 року. Це головний випуск, орієнтований менше на основу основи, а більше на ланцюжок інструментів і на спрощення швидкого переміщення за допомогою Angular у майбутньому, наприклад: оновлення, додавання, кутові елементи, кутовий матеріал + компоненти CDK, кутовий матеріал Компоненти для початківців, робочі місця CLI, підтримка бібліотеки, постачальники трепетних дерев, поліпшення продуктивності анімацій та RxJS v6.

Версія 7

Angular 7 було випущено 18 жовтня 2018 року. Оновлення щодо продуктивності додатків, віртуальної прокрутки, покращеної доступності вибору, тепер підтримує проекцію вмісту за допомогою веб-стандарту для користувацьких елементів та оновлення залежностей щодо Typescript 3.1, RxJS 6.3, Node 10 (все ще підтримує Node 8).

Версія 8

Angular 8 був випущений 28 травня 2019 року. Він містить диференційоване завантаження для всього коду програми, динамічний імпорт для ледачих маршрутів, веб-працівників, підтримку TypeScript 3.4 та Angular Ivy як попередній перегляд. Попередній перегляд кутового плюща включає:

- Створений код, який легше читати та налагоджувати під час виконання
- Швидше час на відновлення
- Покращений розмір корисного навантаження
- Поліпшена перевірка типу шаблону
- Зворотна сумісність
- Майбутні випуски

Один з найголовніших моментів - очікуваний реліз Ivy, сумісного з попереднім, абсолютно нового двигуна візуалізації, заснованого на додатковій архітектурі DOM. Ivy розроблений з урахуванням tree shaking, що означає, що в пакети додатків будуть входити лише частини Angular, які фактично використовуються додатком.

Очікується, що кожна версія буде сумісною з попереднім випуском. Google зобов’язався робити оновлення двічі на рік.

2.2 Spring Boot

Spring Framework забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники та організації, які хочуть створити інформаційну систему, засновану на платформі Java. Через широку функціональність важко визначити найважливіші структурні елементи, з яких він складається. Spring Framework не повністю пов'язана з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності.

Spring Framework - вірогідно, найбільш відомий як джерело розширень (особливостей), необхідних для ефективної розробки складних бізнес-додатків, що не мають важких програмних моделей, які історично були домінуючими в галузі. Ще одна його достоїнство в тому, що він вніс раніше невикористані функціональні можливості в сьогоднішні основні методи розробки, навіть поза платформою Java.

Цей фреймворк пропонує послідовну модель і робить її прикладною для більшості типів додатків, які вже створені на базі платформи Java. Вважаэться, що Spring Framework реалізує модель розробки, засновану на кращих стандартах промисловості, і робить її доступною у багатьох сферах Java.

Spring Framework може бути розглянутий як колекція менших фреймворків або фреймворків у фреймворку. Більшість цих фреймворків може працювати незалежно один від одного, проте, вони забезпечують більшу функціональність при спільному їх використанні.

Ці фреймворки діляться на структурні елементи типових комплексних програм. Схематичне зображення модулей фреймворку на рис.1.5

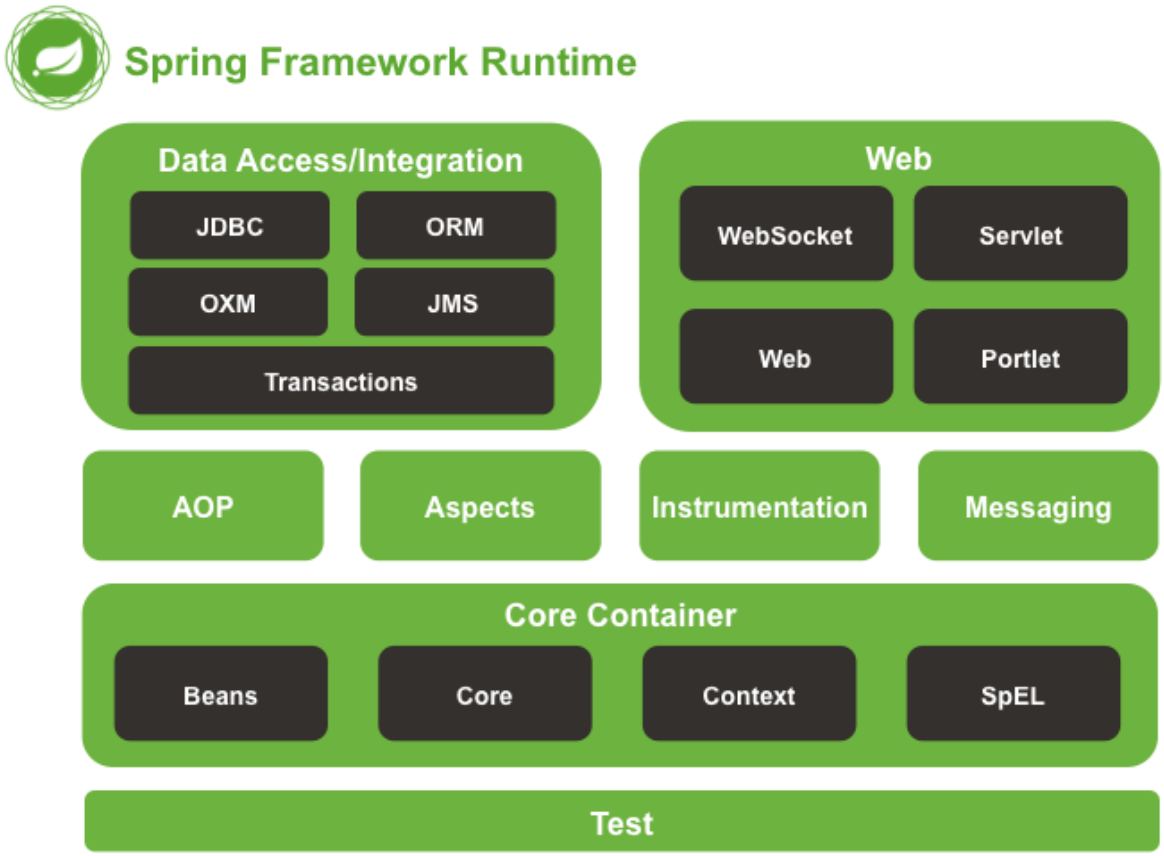

Рис. 2.3 Модулі Spring Fremework

- Inversion of Control контейнер: конфігурація компоненту додатків і управління життєвим циклом Java об'єктів.
- Фреймворк аспектно-орієнтованого програмування: працює з функціональністю, яка не може бути реалізована можливостями об'єктно-орієнтованого програмування на Java без втрат.
- Фреймворк доступу до даних: працює з системами керування базами даних на Java платформі використовуючи JDBC і Object-relational mapping кошти забезпечуючи вирішення завдань, які повторюються в великому числі Java-based environments.
- Фреймворк управління транзакціями: координація різних API керування транзакціями і інструментарій настроюваного управління транзакціями для об'єктів Java.
- Фреймворк Model-view-controller: каркас, заснований на HTTP і

сервлетах надає безліч можливостей для розширення та налаштування (customization).

- Фреймворк віддаленого доступу: конфігурується передача Java-об'єктів через мережу в стилі RPC, підтримуюча RMI, CORBA, HTTP-based протоколи, включаючи web-сервіси (SOAP).
- Фреймворк аутентифікації і авторизації: конфігурується інструментарій процесів аутентифікації і авторизації, що підтримує багато популярних і стали індустріальними стандартами протоколів, інструментів, практик через дочірній проект Spring Security (раніше відомий як Acegi).
- Фреймворк віддаленого управління: конфігурується уявлення і управління Java об'єктами для локальної або віддаленої конфігурації за допомогою JMX.
- Фреймворк роботи з повідомленнями: конфігурується реєстрація об'єктів-слухачів повідомлень для прозорої обробки повідомлень з черги повідомлень за допомогою JMS, поліпшена відправлення повідомлень за стандартом JMS API.
- Тестування: каркас, що підтримує класи для написання модульних і інтеграційних тестів.

Центральною частиною Spring Framework є Inversion of Control контейнер, який надає кошти конфігурації і управління об'єктами Java за допомогою зворотних викликів. Контейнер відповідає за управління життєвим циклом об'єкта: створення об'єктів, виклик методів ініціалізації і конфігурація об'єктів шляхом зв'язування їх між собою[4].

Об'єкти створюються контейнером також називаються Керовані об'єкти або Beans. Зазвичай конфігурація контейнера здійснюється шляхом завантаження XML файлів, що містять Визначення Bean'ов і надають інформацію необхідну для створення bean'ов

Об'єкти можуть бути отримані або за допомогою Пошуку залежності, або Впровадження залежності. Пошук залежності - шаблон проектування, коли викликає об'єкт запитує у об'єкта-контейнера екземпляр об'єкта з певним ім'ям

або певного типу. Впровадження залежності - шаблон проектування, коли контейнер передає екземпляри об'єктів по їх імені іншим об'єктам або за допомогою конструктора, або властивості, або фабричного методу.

Рис.2.4 Логотип сервісу Spring Cloud

Spring Cloud

Eureka Server - це додаток, який містить інформацію про всіх кліентських сервісних додатків.

Каждый мікросервіс зареєстрований на серверах Eureka, і Eureka знає всі кліентські програми, які працюють на кожному порту та IP-адресі. Eureka Server також відомий як Discovery Server.

Аналоги Eureka Server:

- Consul
- Zookeeper
- Cloud Foundry

Якщо прості слова, то - це сервер імен або реестр сервісів. Обязанность - давать ім’я кожному домашньому мікросервісу.

Реєструє мікросервіси і видає їх ір іншим мікросервісом.

Таким чином, кожен сервіс реєструється в Eureka і відправляє ЕНО-запити серверів Eureka, щоб повідомити, що він активний.

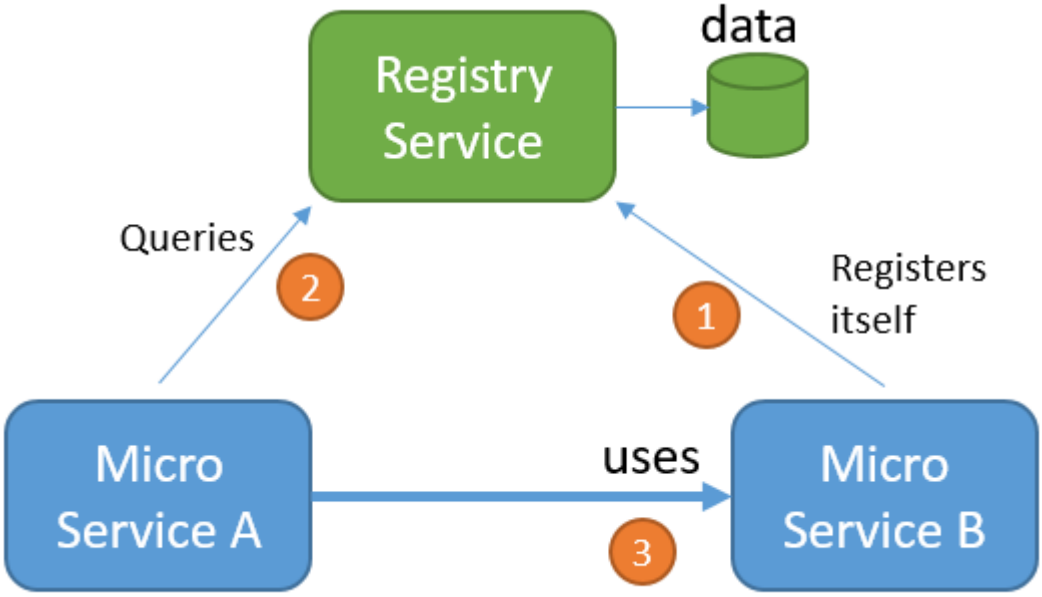

Рис.2.5 Мікросервісна архітектура

- Сервер Eureka

Він містить реєстр служб і API REST, який можна використовувати для реєстрації служб, виданих служб реєстрації та визначення місця розташування інших служб.

- Сервіс Eureka

Будь який додаток, який можна знайти в службі Eureka Server, і може бути виявлено іншими службами. Служба має визначений ідентифікатор (його ще називають VIP), який може посилатися на одного або декількох екземплярів одного і того самого додатка.

- Eureka Instance

Будь який додаток, який зареєстровано на сервері Eureka для виявлення інших додатків.

- Eureka Client

Будь який додаток, можливо, може виявити службу. Він лише запрошує реєстр служб на сервері Eureka, щоб визначити запущені екземпляри мікросервісів.

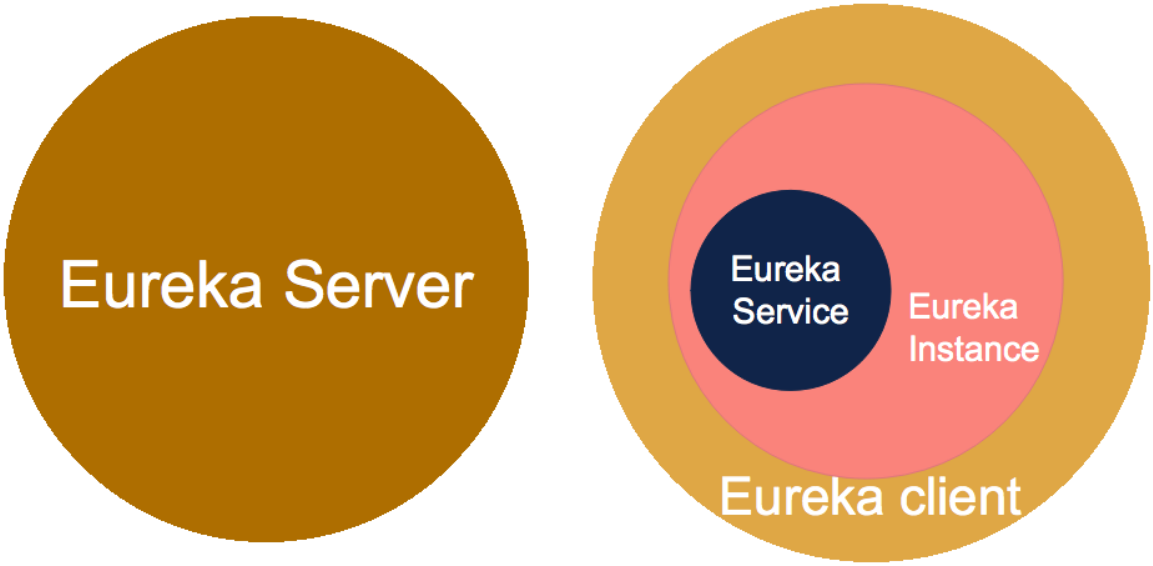

Рис.2.6 Схема Eureka Server.

Додаток може бути, як Eureka Instance так і Eureka Client одночасно, застосовуючи часто, потрібно зробити себе доступними для використання іншими користувачами (щоб вони були екземпляром), і в той час, коли потрібно обов'язково виявити інші служби (щоб вони були клієнтами).

Клієнт Eureka не повинен бути екземпляром Eureka Instance, так як іноді додаток не може нічого запропонувати іншим сервісам .

Інші записи клієнта Eureka зареєстровані на сервері Eureka.

Після цього екземпляр Eureka зареєстрований у сервері Eureka, він також є клієнтом.

Так як Eureka Service пропонує інший API, який може бути відкритий для інших, тому він є екземпляром.

Як це працює?

За замовчуванням клієнт Eureka запускається в стані `STARTING ', що дає можливість отримати екземпляр, щоб виконати ініціалізацію для конкретного додатку, перш ніж він зможе обслуговувати трафік між сервісами.

Клієнт Eureka починає зв'язуватися з серверами Eureka в тій же зоні AWS (за замовчуванням), але якщо він не може знайти сервера, він переключається

на інші зони.

Клієнт Eureka відкидує всі HTTP-з'єднання, які тривать більше 30 секунд, після того як сервер створив з’єднання.

Клієнт взаємодіє з серверами наступним чином:

1) Клієнт Eureka реєструє інформацію про запущений екземпляр на серверах Eureka.

2) Кожні 30 секунд кліент відправляє запит на сервер і інформує сервер про те що екземпляр все ще існує. Якщо сервер не побачив оновлення протягом 90 секунд, він видаляє екземпляр від свого реестра.

3) Клієнт Eureka отримує інформацію про реєстр від сервера та кеширує її у себе локально. І ця інформація оновлюється періодично (кожені 30 секунд), отримуючі оновлення між останніми та поточними додатками. Клієнт автоматично обробляє повторну інформацію.

4) Отримавши оновлення, кліент звіряє інформацію з серверами, перевіряючи кількість екземплярів, і якщо інформація по якій-небудь причині не співпадає, будь-яка інформація витягується з реестра знову.

Клієнт отримує інформацію у форматі JSON, використовуючи клієнт jersey apache.

5) При завершенні роботи кліент відправляє запит про відміну на сервер.

Таким чином, екземпляр видаляється з реестра сервера.

Eureka використовує протокол, який вимагає, щоб клієнти виконували явні дії від реєстрації.

DiscoveryManager.getInstance().ShutdownComponent()

Клієнти, які використали 3 невдалі спроби синхронізації з серверами з інтервалом за 30 секунд, будуть видалені автоматично.

За замовчуванням клієнт Eureka використовує Jersey та Jackson разом із JSON для зв'язку з Eureka Server.

Режим самозберігання:

Сервер Eureka переводить у режим самозбереження, якщо він виявив, що зареєстрована кількість клієнтів, перевищила очікувану кількість одиниць і

первашвши з'єднання для зв'язку з серверами.

Це було зроблено для того, щоб гарантувати те що деякі мережеві програми (велике навантаження або проблеми у мережі) не знищило дані реестра Eureka Server.

Для того, щоб встановити поріг самозбереження, просто потрібно встановити значення у файлі з налаштуваннями:

eureka.renewalPercentThreshold = [0,0, 1,0]

Щоб відключити режим самозбереження, потрібно:

eureka.enableSelfPreservation = false

Як уже згадувалось вище, клієнти намагаються встановити зв'язок із серверами в тій же зоні.

Якщо вони виявляють проблеми при спілкуванню з серверами або якщо сервер не існує в тій же зоні, клієнти переключаються на сервери в іншій зоні.

Взаємодія серверів між собою

Сервери Eureka взаємодіють з іншими, використовуючи цей механізм, який використовується між клієнтами та серверами.

Коли сервер запускається, він вимагає отримувати всю інформацію реестра екземпляра від сусіднього узла.

Якщо при отриманні інформації від вузла виникає проблема, сервер перевіряє всі рівноправні вузли.

У випадку, коли сервер намагається захистити вже наявну у вас інформацію.

Наприклад, може бути сценарій масового відключення, в результаті якого клієнти можуть отримати екземпляри, які більше не існують.

Кращий захист при такому сценарію - швидке відключення та перевірка інших серверів.

Коли сервер запускається і у випадку, коли він не може отримати інформацію з реєстру від сусіднього вузла, він очікує 5 хвилин, щоб клієнти могли зареєструвати свою інформацію.

Якщо ви виникають проблеми у мережі, то між вузлами можна підняти наступні проблеми:

- Ехо-запроси між вузлами, можливо, завершаться невдало, і сервер перейде в режим самозбереження, захищаючи всі поточні стани.

2.3 Amazon Web Services

Amazon Web Services (AWS) є дочірньою компанією Amazon, яка надає платформи та API для хмарних обчислень на замовлення фізичним особам, компаніям та урядам на основі дозованої оплати. У сукупності ці веб-сервіси хмарних обчислень надають набір примітивної абстрактної технічної інфраструктури та розподілених будівельних блоків та інструментів.

Рис.2.7 Amazone Web Services

Один з таких сервісів - Amazon Elastic Compute Cloud, який дозволяє користувачам мати в своєму розпорядженні віртуальну кластерну мережу комп'ютерів, доступну весь час через Інтернет. Версія віртуальних комп’ютерів AWS емулює більшість атрибутів реального комп'ютера, включаючи апаратні центральні процесорні блоки (процесори) та графічні одиниці обробки (GPU) для обробки; локальна / оперативна пам'ять; накопичувач на жорсткому диску / SSD; вибір операційних систем; мережа; попередньо завантажене прикладне програмне забезпечення, таке як веб-сервери, бази даних та управління відносинами з клієнтами (CRM).

Рис.2.8 Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (EC2) є центральною частиною хмарної обчислювальної платформи Amazon.com, Amazon Web Services (AWS), дозволяючи користувачам орендувати віртуальні комп’ютери, на яких можна запускати власні комп'ютерні програми. EC2 заохочує масштабоване розгортання програм, надаючи веб-сервіс, за допомогою якого користувач може завантажувати зображення Amazon Machine (AMI) для налаштування віртуальної машини, яку Amazon називає "екземпляром", що містить будь-яке потрібне програмне забезпечення. Користувач може створювати, запускати та припиняти серверні екземпляри за потребою, оплачуючи другий за активні сервери - звідси і термін "еластичний". EC2 забезпечує користувачам контроль над географічним розташуванням екземплярів, що дозволяє оптимізувати затримки та високі рівні надмірності.

Операційні системи

Після запуску в серпні 2006 року послуга EC2 запропонувала OpenSolaris та Solaris Express Community Edition Linux, а пізніше Sun Microsystems. У жовтні 2008 року EC2 додала операційні системи Windows Server 2003 та Windows Server 2008 до списку доступних операційних систем. У березні 2011 року стали доступними AMI NetBSD. У листопаді 2012 року було додано підтримку Windows Server 2012.

Починаючи з 2006 року, Колін Персіваль, розробник FreeBSD та співробітник служби безпеки, просив Amazon додати FreeBSD. У листопаді 2012 року Amazon офіційно підтримав запуск FreeBSD в EC2. Платформа FreeBSD / EC2 підтримується Percival, який також розробив безпечну дедупликацію Amazon S3-хмарного сервісу резервного копіювання Tarsnap.

Amazon має власний дистрибутив Linux, заснований на Fedora та Red Hat Enterprise Linux, як низька вартість, яка називається AMI Amazon Linux. Версія 2013.03 включена:

- Версія ядра Linux 3.4.34
- Середовище виконання Java OpenJDK (IcedTea6 1.11.4)
- Колекція компілятора GNU gcc.x86_64 4.4.6-3.45.amzn1
- Стійке зберігання

Екземпляр EC2 може бути запущений з вибором двох типів пам’яті для завантажувального диска або «кореневого пристрою». Перший варіант - локальний диск "екземпляр-зберігання" як кореневий пристрій (спочатку єдиний вибір). Другий варіант - використовувати об'єм EBS в якості кореневого пристрою. Об'єм зберігання інстанцій - це тимчасове сховище, яке виживає при перезавантаженні екземпляра EC2, але коли примірник зупиняється або припиняється (наприклад, за допомогою виклику API або через помилку), цей магазин втрачається.

Блок-магазин Amazon Elastic Block Store (EBS) надає необроблені блокові пристрої, які можна приєднати до екземплярів Amazon EC2. Ці блокові пристрої можуть використовуватися, як і будь-які необроблені пристрої блоку. У типовому випадку використання це включатиме форматування пристрою з файловою системою та його встановлення. Крім того, EBS підтримує низку

вдосконалених функцій зберігання, включаючи знімки та клонування. Об'єм EBS може бути розміром до 16 ТБ. Томи EBS побудовані на реплікуванні сховища, так що вихід з ладу одного компонента не призведе до втрати даних. EBS була представлена широкою громадськістю Amazon в серпні 2008 року.

Amazon EBS

Обсяги EBS забезпечують постійне зберігання, незалежно від терміну експлуатації екземпляра EC2, і діють так само, як жорсткі диски на реальному сервері. Точніше, вони виглядають як блокові пристрої операційної системи, які підтримуються дисковими масивами Amazon. ОС може вільно користуватися пристроєм, як хоче. У найпоширенішому випадку завантажується файлова система, а гучність діє як жорсткий диск. Ще одне можливе використання - це створення масивів RAID шляхом комбінування двох або більше томів EBS. RAID дозволяє підвищити швидкість та / або надійність EBS. Користувачі можуть налаштувати та керувати об'ємами пам’яті розмірами від 1 ГБ до 16 ТБ. Томи підтримують знімки, які можна взяти з інструмента GUI або API. Томи EBS можна приєднувати або від'єднувати від екземплярів під час їх запуску та переміщувати з одного примірника в інший.

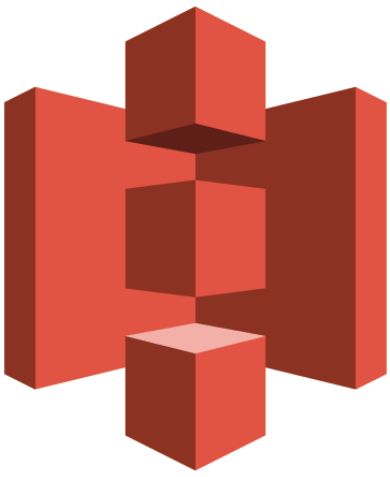

Рис.2.9 Amazon Elastic S3 bucket

Проста служба зберігання даних (S3) - це система зберігання даних, в якій дані доступні для екземплярів EC2 або безпосередньо по мережі для відповідних аутентифікованих абонентів (вся комунікація здійснюється через HTTP). Amazon не стягує плату за пропускну здатність для зв'язку між екземплярами EC2 та сховищем S3 "в одному регіоні". Доступ до даних S3, що зберігаються в іншому регіоні (наприклад, до даних, що зберігаються в Європі від екземпляра EC2 East Coast США), буде виставлено рахунок за звичайними тарифами Amazon.

Зберігання на базі S3 - ціна за гігабайт на місяць. Програми отримують доступ до S3 через API. Наприклад, Apache Hadoop підтримує спеціальну файлову систему s3: для підтримки зчитування та запису в сховище S3 під час завдання MapReduce. Існують також файлові системи S3 для Linux, які монтують віддалений файл зберігання файлів S3 на зображення EC2, ніби це локальне сховище. Оскільки S3 не є повноцінною файловою системою POSIX, речі можуть поводитись не так, як на локальному диску (наприклад, відсутність підтримки блокування).

Amazon Elastic IP

Еластична функція IP-адреси Amazon схожа на статичну IP-адресу в традиційних центрах обробки даних, з однією ключовою відмінністю. Користувач може програмно відображати еластичну IP-адресу на будь-якому екземплярі віртуальної машини без допомоги адміністратора мережі та без необхідності чекати, коли DNS поширить прив'язку. У цьому сенсі Elastic IP Address належить до облікового запису, а не до екземпляра віртуальної машини. Він існує, поки він не буде видалено явно і залишається пов'язаним з обліковим записом навіть тоді, коли він пов’язаний із жодним примірником.

2.4 Предметна область Web- додатку car-dealer

Основною функцією веб-додатку розміщення оголошень від приватних

осіб та компаній які займаються продажом та купівлею автомобілей. Веб-додаток буде функціювати як віртуальний торговельна площадка де у відкритому доступі розміщуються оголошення про продаж і характеристики автомобілів.

Стурктура веб-додатку:

- Каталог машин;
- Пошук товарів для автож
- Повнотекстовий пошук автомобілей;
- Детальний пошук, за типом машини, рік випуску, ціна і тд.
- Реєстрація нових користувачів;
- Авторизація користувачів;

У веб-додатків є чимало переваг у порівнянні зі звичайними десктоп-застосуноками . Наприклад, розробка веб-додатків забезпечує повну сумісність при роботі на різних платформах. Якщо звичайне оффлайн-додаток доводиться «заточувати» під певні системні вимоги конкретної платформи, то веб-додаток завжди і всюди може бути доступним для будь-якої локальної машини. Все що потрібно - це браузер і включений доступ до інтернету.

Всі сучасні веб-додатки мають важливі властивості, серед яких особливо велике значення мають:

- масштабованість веб-системи;
- інтеграція з іншими системами;
- розмежування прав доступу до різного функціоналу;
- зручне розгортання і обслуговування системи.

Наявність подібних властивостей дозволяє використовувати веб-додатки і веб-системи максимально ефективно і зручно. Наприклад, завдяки властивості масштабованості можна без внесення кардинальних змін розширювати веб-систему для роботи з постійно зростаючим числом користувачів, додавати в неї нові функції. Розробка веб-додатків надає широкі можливості по створенню для компаній багатофункціональних онлайн-інструментів для оптимізації або

вирішення різних бізнес-задач.

Веб-додаток car-dealer, буде заснований на сучасних базах даних RDS, з програмно-технічним комплексом, і буде призначений для зручного пошуку та продажу нових та вживаних автомобілей.

Завданням системи є:

1) Забезпечети безпеку обробки та зберігання персональних данних користувача;

2) Облік користувачів;

3) Облік колонок для карток. Колонки повинні забепечити сортування карток за пріорітетом;

4) Облік карток. Користувач має можливість додавати до картки: опис завдання, прикріплення у вигляді посилання, пріорітет виконання цієї картки;

Проаналізувавши предметну область, можна сказати, що розробка даного веб-додатку є вельми актуальною і дозволить користувачам зручно керувати своїми оголошеннями та зв’язуватись з іншими коричстувачами.

ВИСНОВОК ДО РОЗДІЛУ 2

На сьогоднішній день існує безліч технологій та програмних засобів для реалізації найсміливіших ідей будь-якого розробника. До основних переваг Spring відносяться: простота; зручність тестування; використання Spring позитивно відображається на дизайні застосування і простоті коду.

У другому розділ проведено аналіз предметної області для web-додатку. Веб додаток дозволить користувачам розміщувати оголошення від приватних осіб та компаній які займаються продажом та купівлею автомобілей. Виділено основні вимоги до web-додатку які відповідать сучасним правилам для стоверення програмного забезпечення.

РОЗДІЛ 3

РОЗРОБКА WEB-ДОДАТКУ CAR-DEALER

3.1 Проектування та створення бази даних

Після проведення аналізу предметної області web-додатку необхідно побудувати її концептуальну схему. Необхідно виділити основну архітектуру

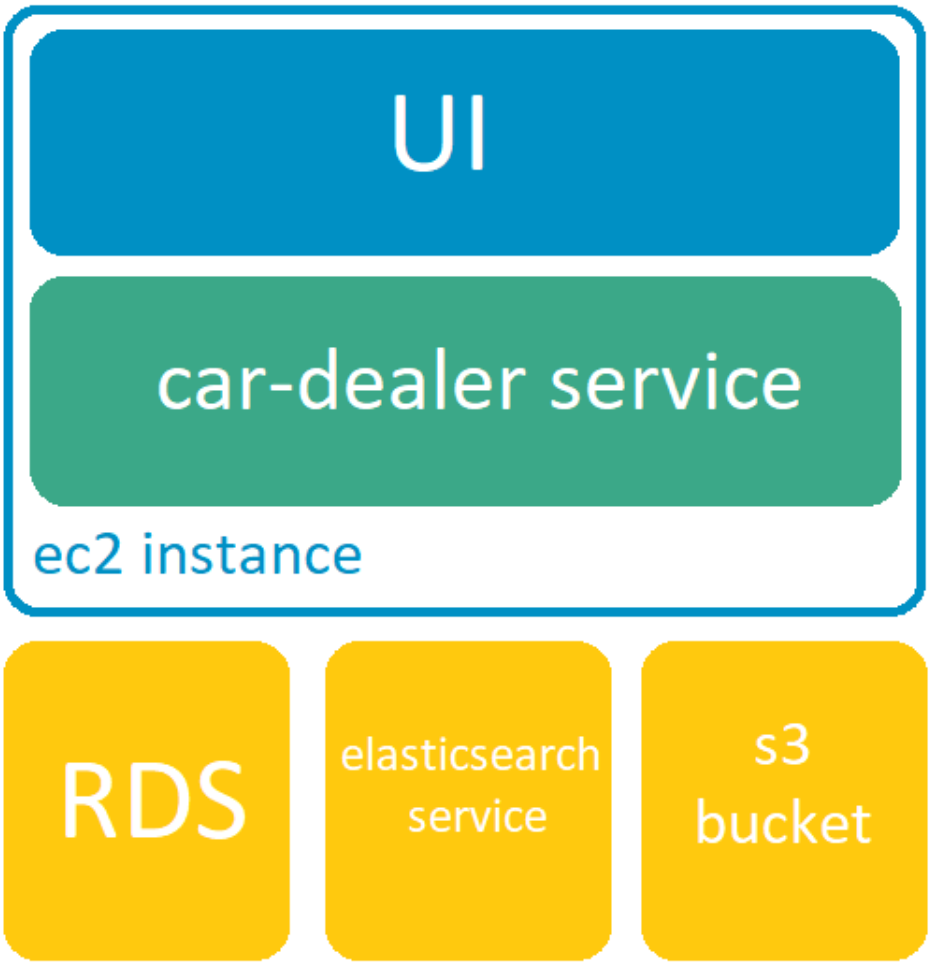

Рис.3.1 Архітектура серверу для веб-додатку car-dealer

1) Ec2 instance – веб сервер з опереційно системою Centos 7;

2) UI додаток відповідає за інтерфейс для користувачів (фрондент);

3) Car-dealer service – основна бізнес-логіка веб-додатку (бекенд);

4) RDS – реляційна база данних як окремий сервіс;

5) Elasticsearch service – сервіс для повно текствого пошуку;

6) S3 bucket – сервіс для зберігання файлів, тобто фотографій автомобілей;

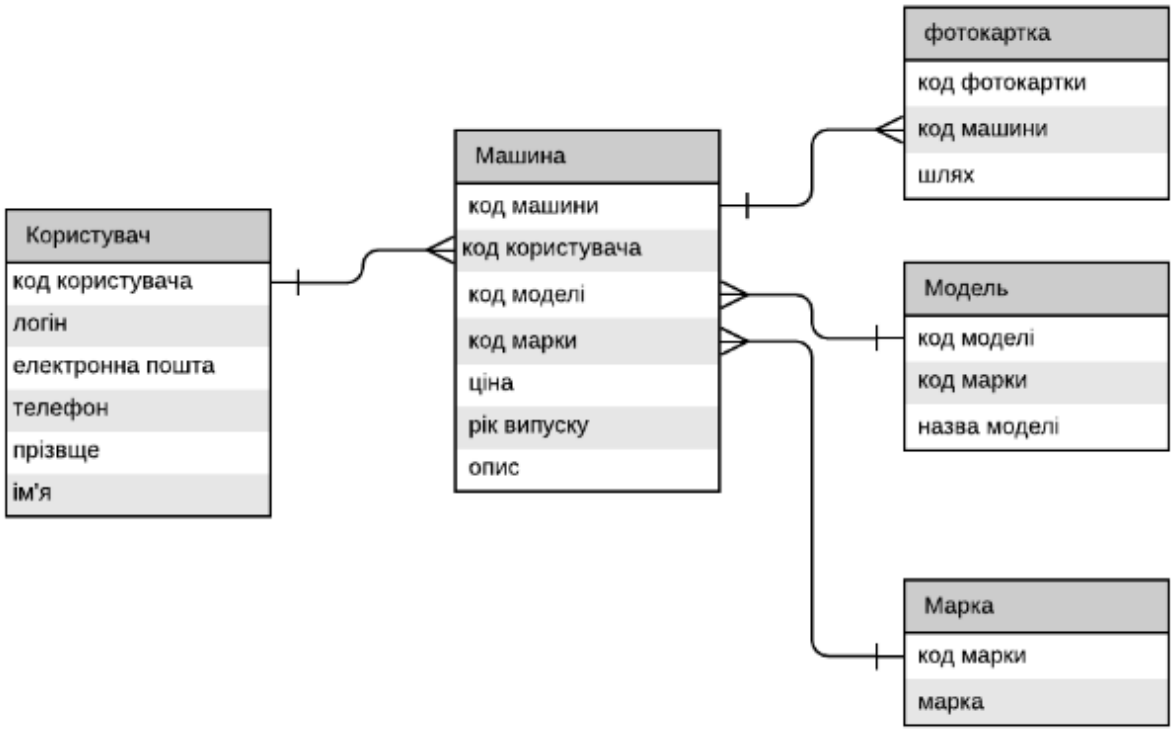

Рис. 3.2 Початкова модель БД

Схема реляційних відношень називається коректною, якщо в ній відсутні небажані функціональні залежності. Виправити некоректні схеми можна за допомоги декомпозиції (розкладання) реляційних відношень на множину реляційних відношень, які не містять некоректностей. Це і є суттю процесу нормалізації. Е. Код визначив 3 рівні нормалізації: 1НФ, 2НФ, 3НФ.

Реляційне відношення перебуває в 1НФ, якщо всі його атрибути – прості, тобто кожен атрибут повинен містити лише одне значення.

Реляційне відношення перебуває у 2НФ, якщо воно перебуває у 1НФ і

всі його не ключові атрибути функціонально повно залежать від ключа.

Реляційне відношення перебуває у 3НФ, якщо воно перебуває в 2НФ і не містить транзитивних залежностей неперервних атрибутів від можливих ключів.[6]

Після проведення нормалізації реляційних залежностей виділено сім основних сутностей з атрибутами:

1) Роль (код ролі, роль);
2) Користувач (код користувача, логін, пароль, ім’я, прізвище);
3) Роль користувача (код ролі, код користувача);
4) Контакти користувача(код котакту, тип, значення, еод користувача)
5) Оголошення (код оголошення, код користувача, код машини, код головної фотокартки);
6) Машина (код машини, код моделі, ціна, код валюти, рік випуску, опис);
7) Фотокартка (код фотокартки, шлях фотокартки з s3);
8) Марка (код марки, марка);
9) Модель (код моделі, код марки, назва моделі);
10) Валюта (код валюти, назва, курс нбу);
11) Коментарій (код коментаря, код користувача, коментарь, код оголошення);

Таким чином щоб користувач мав декілька ролей, використовуючи з’єдння багато до багатьох була створенна третя таблиця «Роль користувача». Користувач може мати декілька оголошень, але оголошення може мати лише одного користувача, а також декільяка контактів. Сутність оглошення може мати тільки одну машину та дакілька комантарів. Оголашення може мати тільки одну головну фотокарту та декілька другорядних фотокарток.

Використання CASE-засобу AllFusionERwinDataModelerbyCA надає можливість експортувати схеми функціональних залежностей в фізичні моделі баз даних. Для цього необхідно перемкнутися з логічного проектування на фізичне і відредагувати вже створену схему таким чином, щоб вона повністю

відображала структуру бази даних.

Таблиця role (Рис. 3.3.) буде містити інформацію про ролі в web-додатку:

role_id – унікальний ідентифікатор ролі(первинний ключ), тип даних – ціле число (***INT***);

role – назва ролі, тип даних – символьний(***VARCHAR***);

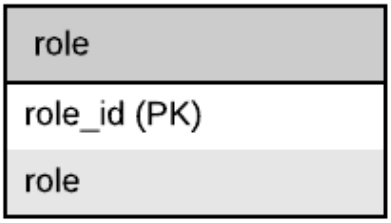

Рис. 3.3 Графічне представлення таблиці role

Таблиця user (Рис. 3.4) буде містити інформацію про користувача в web-додатку:

user_id – унікальний ідентифікатор користувача(первинний ключ), тип даних – ціле число (***INT***);

login – логін користувача, тип даних – символьний(***VARCHAR***), повинен бути унікальним для кожно користувача;

password – пароль користувача, тип даних – символьний(***VARCHAR***);

first_name – ім’я користувача, тип даних – символьний(***VARCHAR***);

second_name – прізвище користувача, тип даних – символьний(***VARCHAR***);

Для всіх встановлюється *NOTNULL*.

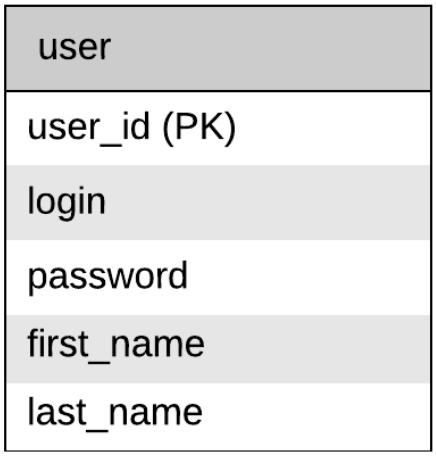

Рис. 3.4 Графічне представлення таблиці user

Таблиця user_role (Рис. 3.5) буде містити інформацію про ролі користувачів в web-додатку:

user_id – унікальний ідентифікатор користувача(зовнішній ключ), тип даних – ціле число (***INT***);

role_id – унікальний ідентифікатор ролі(зовнішній ключ), тип даних – ціле число (***INT***);

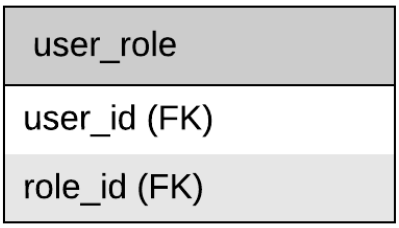

Рис. 3.5 Графічне представлення таблиці user_role

Таблиця advert (Рис. 3.6) буде містити інформацію про оголошення:

advert_id – унікальний ідентифікатор оголошення(первинний ключ), тип даних – ціле число (***INT***);

user_id – унікальний ідентифікатор користувача якому належить оголошення (зовнішній ключ), тип даних – ціле число (***INT***);

car_id – унікальний ідентифікатор машини для оголошення (зовнішній ключ), тип даних – ціле число (***INT***);

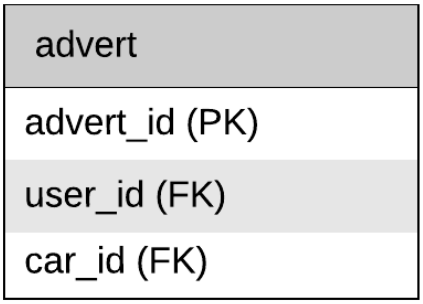

Рис. 3.6 Графічне представлення таблиці adver

Таблиця car (Рис. 3.7) буде містити інформацію про машину для оголошення:

car_id – унікальний ідентифікатор машини (первинний ключ), тип даних – ціле число (***INT***);

advert_id – унікальний ідентифікатор оголошення (зовнішній ключ), тип даних – ціле число (***INT***);

model_id – унікальний ідентифікатор моделі (зовнішній ключ), тип даних – ціле число (***INT***);

currency_id – індетифікатор валюти (зовнішній ключ), тип даних – символьний(***VARCHAR***);

price – ціна автомобіля, тип даних – ціле число (***INT***);

year – назва списку, тип даних – ціле число (***INT***);

description – опис машини, тип даних – символьний(***VARCHAR***);

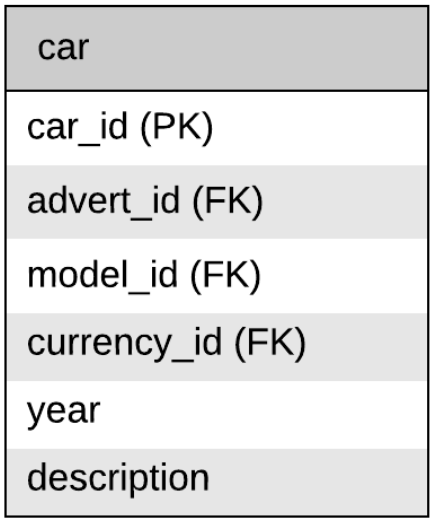

Рис. 3.7 Графічне представлення таблиці car

Таблиця currency (Рис. 3.8) буде містити інформацію про валюту:

currency_id – унікальний ідентифікатор валюти (первинний ключ), тип даних – ціле число (***INT***);

currency – назва валюти, тип даних – символьний(***VARCHAR***);

exchange_rate – курс валюти, тип даних – символьний(***VARCHAR***);

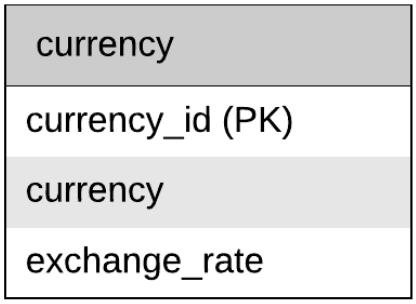

Рис. 3.8 Графічне представлення таблиці currency

Таблиця brand (Рис. 3.9) буде містити інформацію про виробників автомобілей:

brand_id – унікальний ідентифікатор виробника (первинний ключ), тип даних – ціле число (***INT***);

brand – назва виробника автомобіля, тип даних – символьний(***VARCHAR***);

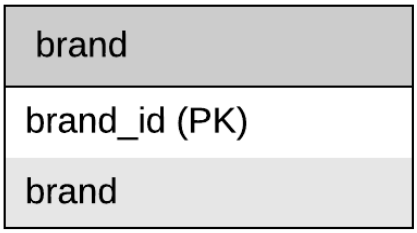

Рис. 3.9 Графічне представлення таблиці brand

Таблиця brand (Рис. 3.10) буде містити інформацію про саме модель виробника:

model_id – унікальний ідентифікатор моделі (первинний ключ), тип

даних – ціле число (***INT***);

model_id – унікальний ідентифікатор виробника (зовнішній ключ), тип даних – ціле число (***INT***);

model – назва моделі автомобіля, тип даних – символьний(***VARCHAR***);

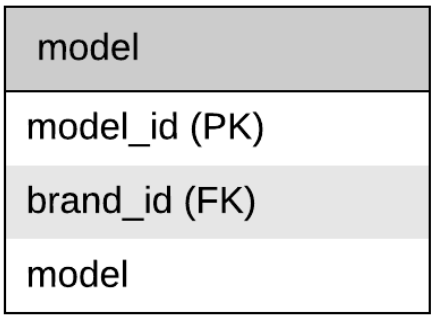

Рис. 3.10 Графічне представлення таблиці model

Таблиця photo (Рис. 3.11) буде містити інформацію фотографію для оголошення:

photo_id – унікальний ідентифікатор фотокартки (первинний ключ), тип даних – ціле число (***INT***);

advert_id – унікальний ідентифікатор оголошення (зовнішній ключ), тип даних – ціле число (***INT***);

path – шлях фотографії у сховищі S3 Bucket, тип даних – символьний(***VARCHAR***);

main – значення головної фотокартки тип даних – ціле число (***BOOLEAN***);

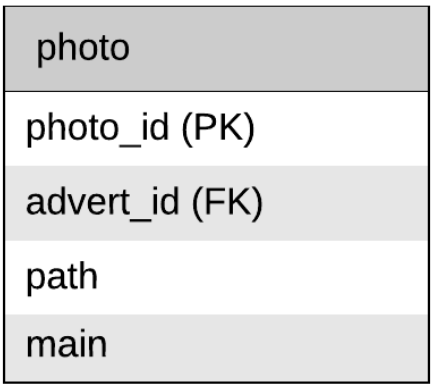

Рис. 3.11 Графічне представлення таблиці photo

Таблиця (Рис. 3.12) буде містити інформацію про кометарій від інших користувачів під оголошенням:

photo_id – унікальний ідентифікатор коментаря (первинний ключ), тип даних – ціле число (***INT***);

advert_id – унікальний ідентифікатор оголошення (зовнішній ключ), тип даних – ціле число (***INT***);

user_id – унікальний ідентифікатор користувача якому належить оголошення (зовнішній ключ), тип даних – ціле число (***INT***);

value – текстове значення коментаря від користувача (***VARCHAR***);

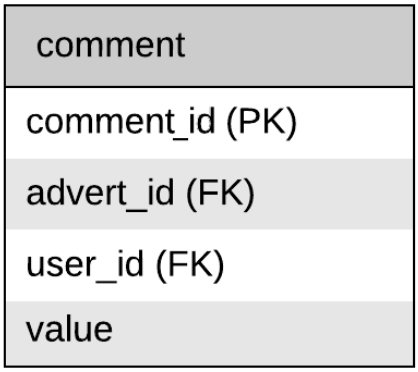

Рис. 3.12 Графічне представлення таблиці comment

Таблиця contact (Рис. 3.6) буде містити інформацію про контакти користувачів:

contact_id – унікальний ідентифікатор оголошення (первинний ключ), тип даних – ціле число (***INT***);

user_id – унікальний ідентифікатор користувача якому належить оголошення (зовнішній ключ), тип даних – ціле число (***INT***);

type – тип контакту (телефон, пошта і тд) (***VARCHAR***);

value – текстове значення контакту користувача (***VARCHAR***);

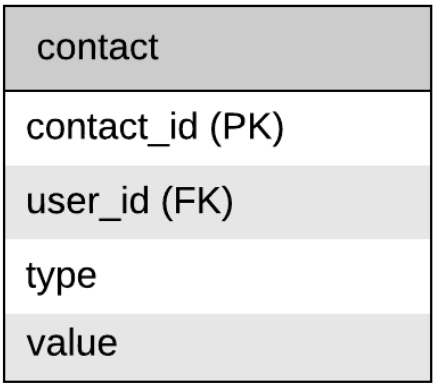

Рис. 3.13 Графічне представлення таблиці contact

Після редагування всіх таблиць схема структури бази даних набуває остаточного вигляду(Рис. 3.10).

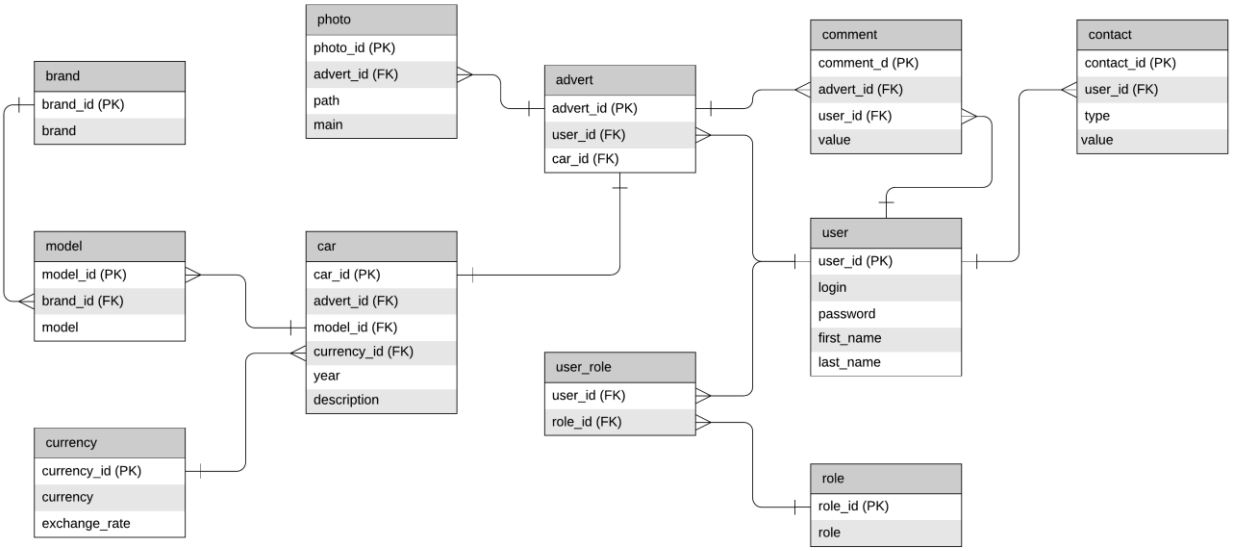

Рис. 3.14 Фізична модель бази даних

Далі розроблену схему необхідно представити у вигляді SQL-сценарію. Для цього використовується Export → SQL в налаштуваннях якого вказана цільова база даних MySQL.

Після натискання кнопки Generate SQL необхідно завантажити файл *SQL*-сценарію в якому є всі необхідні *SQL*-запити для створення БД(текст данного файлу наведено в Додатку А)

Для створення БД використовуючи середовище розробики Intellij IDEA

необхідно запустити на виконання збережений до цього файл *SQL*-сценарію.

3.2 Реалізація web-додатку task list

Серверне налаштування web-додатка

Для реалізації проекту task-list були обрані такі технології:

- Spring Boot. Використовується для запуску серверу додатків;
- Spring Security. Використовується для забезпечення авторизації і аутетифікації користувача в системі.
- MySQL Connector – забезпечує зв’язок бекенду з БД.
- Spring Data – ORM фреймворк, який зв’язує таблиці БД з сутностями
- Apache Tomcat – контейнер сервлетів, забезпечує компіляцію представлення для фронтенду;
- Java Servlet – стандартизований API для створення динамічного контенту до веб-сервера;
- Junit – бібліотека для тестування програмного забезпечення для мови Java;

Для побудови та підключення всіх залежностей які прераховані вище, було обрано технологію «Apache Maven». Це засіб автоматизації роботи з програмними проектами. Використовується для управління (management) та складання (build) програм. Maven конфігурує проекти за допомогою конструкції ProjectObjectModel, що зберігається в файлі POM.xml. Для підключення потрібних елементів з центрального репозиторія https://search.maven.org/ потрібно знайти необхідну технологію та скопіювати залежність. Пошук технології забражено на рис.3.11

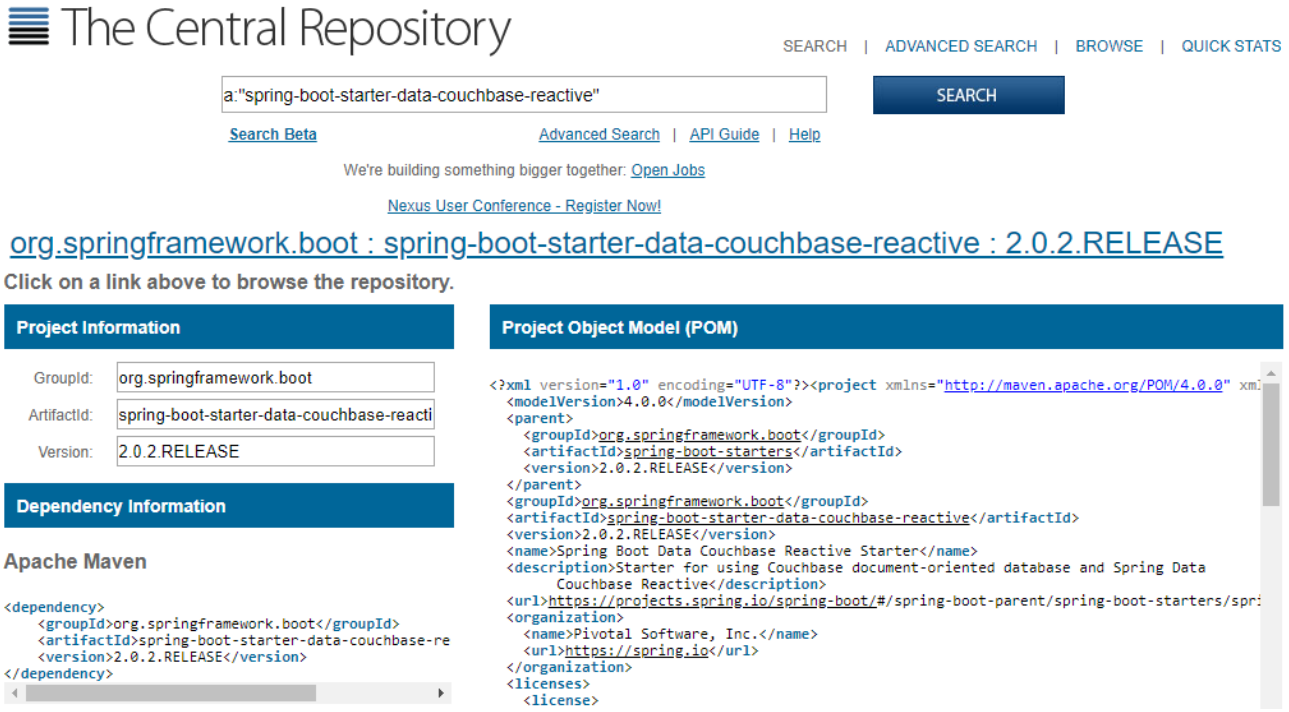

Рис. 3.15 Пошук технології

Код остаточного файлу POM.xml наведенно в додатку Б. Для компіляції проекту в терміналі треба виконати команду *mvn clean*, під час виконання maven завантажує всі залежності до локального репозиторію.

Після успішного виконання команди необхідно сконфігурувати файл налаштування *application.properies* (Код представленно в додатку В). В данному файлі вказується назва протоколу з'єднання(*jdbc:mysql*), хост і порт підключення(*localhost:3306*), ім'я бази даних(*task-list*), ім'я користувача та пароль(*root, mysql).*

Для того щоб зупустути сервер додатків було створенно головний клас *SpringBootApp.java*. Аннотація *@SpringBootApplication* допомагає *Spring*-контейнеру розпізнати цей клас. В класі об`явленний метод main(), котрий створює окремий потік JVM. Він і є головною точкою запуску web-додатку.

Щоб запустити web-додаток треба виконати команду *mvn spring-boot:run*. Web-додаток запуститься на локальному сервері комп`ютера і буде займати порт № 8080. Строка *Tomcat started on port(s): 8080 (http): Started SpringBootApp in 20.948 seconds (JVM running for 61.987)* в терміналі повідомляє про те що додаток успішно запустився.

Забезпечення авторизації та аунтентифікації

Spring Security це Java / Java EE фреймворк, що надає механізми побудови аутентифікації та авторизації, а також інші можливості забезпечення безпеки для промислових прикладних програм, створених за допомогою Spring Framework.

Клас *SecurityConfiguration.java* створенний для забезпечення механізму аутентифікації та авторизації. Аннотації @Configuration, @EnableWebSecurity вказує на те що данний клас буде служити налаштуванням для цього фреймворку.

За специфікацією Spring Security, клас-налаштуваня повинен наслідуватись від класу *WebSecurityConfigurerAdapter.java* і первизначити метод *configure()*. Цей метод створює сессію для кожного авторизованого користувача, виконуючи запити до БД перевіряє логін та пароль за допомогою поля *dataSource* данного класу. Поля *usersQuery* та *rolesQuery* це запити в БД, які об`єкт *dataSource* використовує для перевірки. (Код данного класу наведенно в додатку Г).

Реалізація моделей для web-додатку car-dealer

Після створення БД всі сутності потрібно відобразити у вигляді java класів. Було виділено такі об`єкти як : користувач, роль користувача, дошка, список задач, завдання, прикріплення.

Java Persistence API (JPA) - специфікація API Java EE, надає можливість зберігати в зручному вигляді Java-об'єкти в базі даних. Існує кілька реалізацій цього інтерфейсу, одна з найпопулярніших використовує для цього Hibernate, Eclipse Link, Spring Data. JPA реалізує концепцію ORM.

Entity (сутність/утворення) — об'єкт для якого забезпечується ORM. Класи Entity задаються аннотацією *@Entity* або перелічуються у XML дескрипторі. Клас Entity повинен мати конструктор без аргументів, з рівнем доступу — *public* або *protected*. Якщо сутність передається як віддалений об'єкт

(*remote object*), вона має реалізувати інтерфейс *Serializable*. Клас *Entity* не може бути завершеним (*final*) або мати завершені методи.

Поля класу повинні відповідати колонкам сутності з БД. Аннотації *@Entity* та *@Table(name = “user”)* вказують на те що данний клас є відображенням таблиці в БД. В аннотацію *@Table* передається атрибут *name* значення якого повинно відповідати назві таблиці користувача.

Для поля з унікальним індитифікатором потрібно присвоїти аннотації @Id та @GeneretedValue. В аннотацію *@GeneretedValue* передається атрибут *strategy* , існує декілька констант для атрибуту, а саме *AUTO, INDENTITY, SEQUENCE.*

Сутності в БД мають зв`язки для забезпечення зв`язків на стороні java в специфікації JPA виділенно три види аннотацій:

- *@OneToOne* – Визначає однозначну асоціацію іншому об'єкту, який має кратність "один до одного".
- *@OneToMany* Визначає багатозначну асоціацію з одного до багатьох об`єктів.
- *@ManyToMany* Визначає багатозначну асоціацію з багатьох до багатьох об`єктів.

Приклад реалізації класу *Car.java* наведенно на рис 3.12. Поля id, name, класу відповідають сутності Списку задачю. Так як канбан-дошка може складатись з декількох списків, було додано поле *board* з аннотаціями *@ManyToOne* та *@JoinColumn* з атрибутом *name* який повинен відповідати зовнішньому ключу в БД. Поле *task* описує сукупніть завданнь які належать об’єкту *Car.*

```java
package com.nau.icit.model;

import javax.persistence.*;

@Entity
@Table(name = "car")
public class Car {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "board_id")
    private Long id;

    @OneToOne
    @JoinColumn(name = "advert_id")
    private Advert advert;

    @OneToOne
    @JoinColumn(name = "model_id")
    private Model model;

    @OneToOne
    @JoinColumn(name = "currency_id")
    private Currency currency;

    @Column(name = "price")
    private String price;

    @Column(name = "price")
    private String year;
```

Рис. 3.16 Клас-сутність *Car*

Сутності: користувач, роль, ролі користувача, фотокартка, коментарь, машина, валюта, модель, виробник, наведенно в додатку Д.

CRUD операції для моделей

CRUD — (англ. create read update delete) 4 базові функції управління даними «створення, зчитування, зміна і видалення».

Термін «CRUD» також вживають стосовно інтерфейсу користувача. Для прикладу, в web-додатку task list, один із базових об'єктів — це запис про завдання. Як мінімум, програма повинна надавати корисувачу функції для:

- Додавання оголошень;
- Пошук і зчитування оголошень;
- Редагування оголошень,

- Видалення існуючих оголошень.

Без цих базових операцій програма не може вважатися придатною до користування.

Для створення цих операцій було обрана технологія Spring Data. Вона забезпечує підтримку репозиторіїв для Java Persistence API (JPA). Це полегшує розробку додатків, які потребують доступу до джерел даних JPA. Мета абстракції даних полягає у суттєвому зменшенні кількості коду, необхідного для введення шарів доступу до даних.

Центральний інтерфейс у абстрактному репозиторію Spring Data – це інтерфейс Repository. Цей інтерфейс діє, насамперед, як маркерний інтерфейс для захоплення типів данних, а також допомагає виявити інтерфейс для моделі, який і створить реалізацію CRUD операцій. Типові операції описувати не потрібно такі як збрерігання, видалення, редагування. Якщо ж розробнику потрібен специфічний метод, то він має вказати в назві методу ключові слова, наприклад *deleteById* буде видяти сутність за індифікатором. Розробнику не потрібно реалізовувати тіло методу, *Spring*-контейнер зробить це за нього.

Інтерфейс JpaRepository є насілідником батьківського інтерфейсу Repository. Тому для опису CRUD операцій необхідно наслідуватись від JpaRepository, так як він є узагальненим класом йому потрібно передати два аргументи це модель та тип унікального індтифікатора який використовується в данній моделі.

Інтерфейс UserRepository буде описувати CRUD операції для сутності Користувач. Методи інтерфейсу:

- *User findUserByLogin(String login)* на вхід приймає логін користувача та знаходить відповідний запис в БД і повертає об`єкт користувача, якщо запис не знайдено повертає посилання на *null*.

Інтерфейс RoleRepository буде описувати CRUD операції для сутності Роль користувача. Методи інтерфейсу:

- Role findByRole (String role) на вхід приймає назву ролі та знаходить відповідний запис в БД і повертає об`єкт ролі, якщо запис не знайдено повертає

посилання на null.

Інтерфейс CarRepository буде описувати CRUD операції для сутності Дошка. Методи інтерфейсу:

- *Car findBoardById (Long id)* на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт машини, якщо запис не знайдено повертає посилання на *null*;
- *List<Car> findCarsByUser (User user)* на вхід приймає об`єкт користувача та знаходить всіх машин які належать користувачу і повертає список машин, якщо записів не знайдено повертає посилання на *null*.

Інтерфейс AdvertRepository буде описувати CRUD операції для сутності Списоок задач. Методи інтерфейсу:

- *Adver findAdvertListById (Long id)* на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт дошки, якщо запис не знайдено повертає посилання на *null*;
- *List<Advert> findAdvertsByBoardId (Long id)* на вхід приймає, індитифікатор користувача та знаходить всі оголошення машин які належать данномукористувачу, якщо записів не знайдено повертає посилання на *null*;
- *void deleteAllByUser (User user)* на вхід приймає об`єкт користувача та видаляє всі списки задач які налажали данному користувачу.

Інтерфейс ContactRepository буде описувати CRUD операції для сутності Завдання. Методи інтерфейсу:

- *Contact findContactById (Long id)* на вхід приймає індитифікатор та знаходить відповідний запис в БД і повертає об`єкт завдання, якщо запис не знайдено повертає посилання на *null*;
- *void deleteAllByAdvert (Advert advert)* на вхід приймає об`єкт оголошення та видаляє всі контакти які були прікріплені та налажали данному оголошенню;

Інтерфейс PhotoRepository буде описувати CRUD операції для сутності Прикріплення. Методи інтерфейсу:

- *Photo findPhotoById (Long id)* на вхід приймає індитифікатор та

знаходить відповідний запис в БД і повертає об`єкт прикріплення, якщо запис не знайдено повертає посилання на *null*;

- *void deleteAllByAdvert (Advert advert)* на вхід приймає об`єкт завдання та видаляє всі прикірплення які налажали данному оголошенню.

Код інтерфейсів наведенно в додатку Е.

Опис контроллерів для web-додатку

Контроллер – це такий клас який приймає HTTP запит користувача, опрацьовує його і повертає сторінку представлення в браузер. Аннотація @Controller є обов`язковою для класу, при запуску додатка Spring-контейнер стоворить об`єкт цього класу, і потім буде опрацьовувати запити зі сторони клієнта.

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів.

Контроллер LoginController має такі методи:

- метод login() повертає сторінку аутентифікації;
- метод registration() приймає запит на реєстрацію нового користувача, превіряє правильність введенних данних. Повертає сторінку аутентифікаціїї.

Контроллер BoardListController має такі методи:

- метод *getBoards()* приймає запит коли користувач заходить на сторінку з дошками. Метод виконує запит до БД, знаходить всі дошки які належать користувачу. Генерує сторінку з дошками та повертає цю сторінку клієнту.
- метод *saveBoard()* приймає запит коли користувач зберігає нову дошку. Метод звертається до сессії в якій користувач зареєструвався та зберігає нову дошку. Далі генерує нові сторінку і повертає клієнту.
- метод *removeBoard()* приймає запит з індетифікатором від

користувача на видаленя дошки. Знаходить відповідну дошку і видаляє дошку та всі записи які поязані з нею.

Контроллер AdvertController має такі методи:

- метод getAdvert() приймає запит з індетифікатором дошки від користувача. Звертається до БД, знаходить списки задач, завдання. Виконує сортування завдань за пріорітетом. Генерує сторінку дошки і повертає її клієнту.
- метод editAdvert() приймає запит на редагування дошки. Викононує оновлення запису в БД і повертає оновленну сторінку.
- метод addPhotos() приймає запит коли користувач зберігає фотографії для оголошення. Метод звертається до сессії в якій користувач зареєструвався та зберігає нові фотографії.
- метод editTaskList() приймає запит на редагування назви списку задач. Викононує оновлення запису в БД і повертає оновленну сторінку.
- метод removeAdvert() приймає запит на видалення списку задач. Видаляє відповідний запис в БД і повертає оновленну сторінку.
- метод addTask() приймає запит коли користувач зберігає нове завдання. Метод звертається до сессії в якій користувач зареєструвався та зберігає нове завдання.

Контроллер TaskController має такі методи:

- методи getTask() примає запит з індетифікаторм завдання. Виконує пошук відповідного завдання та прикріпленнь які належать завданню. Генерує сторінку і повертає клієнту.
- Метод editTask() обробляє запит користувача на редагування завдання. Викононує оновлення запису в БД і повертає оновленну сторінку.

Код контроллерів наведенно в додатку Є.

Опис тестів для web-додатку

Для тестування web-додатку була обрана технологія Junit. JUnit — бібліотека для тестування програмного забезпечення для мови Java. Процес

технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення баґів (помилок або інших дефектів).

Для web-додатку було реалізовано 13 тестів. Тестування повинно проводитись тільки на сервісні класи, а саме на контроллери.

Опис тестування контроллеру авторизації.

Коли користувач заходить на сторінку авторизації він повинен ввести свої данні. Для тестування цього контроллеру потрібно штучно створити об`єкт користувача. Поля цього об`єкту ми зазделегіть знаємо. В методі *setup()* потрібно сворити об`єкт і заповнити данними. Знаючи який користувач буде виконувати запит і відповідно результат відповіді теж відомий. Залишається тільки виконати тестування, якщо тест виконанний успішно то і сторінка авторизації працює правильно.

Тести для web-додатку наведенно і додатку Ж.

3.3 Web-додаток car-dealer

Web-додаток car-dealer складається з 6 сторінок: головної сторінки, сторінки для реєстрації, сторінка для авторизації, сторінка оголошень, сторінка додавання оголошення. Розглянемо детально усі сторінки окремо:

- Головна сторінка

Головна сторінка уявляє собою опис web-додаткв, на якій користувач має змогу прочитати і дізнатися про можливості додатка, а також перейти на інші сторінки web-додатку.

На сторінці реєстрації(рис. 3.14) користувач має змогу створити власний

аккаунт в системі. Користувач повинен ввести данні про ім`я, прізвище, електронну пошту, пароль, логін. Якщо усі данні введено у вірному форматі, його аккаунт буде створено і його переадресує на сторінку авторизаціїї з повідомленням про успішне створення облікового запису.

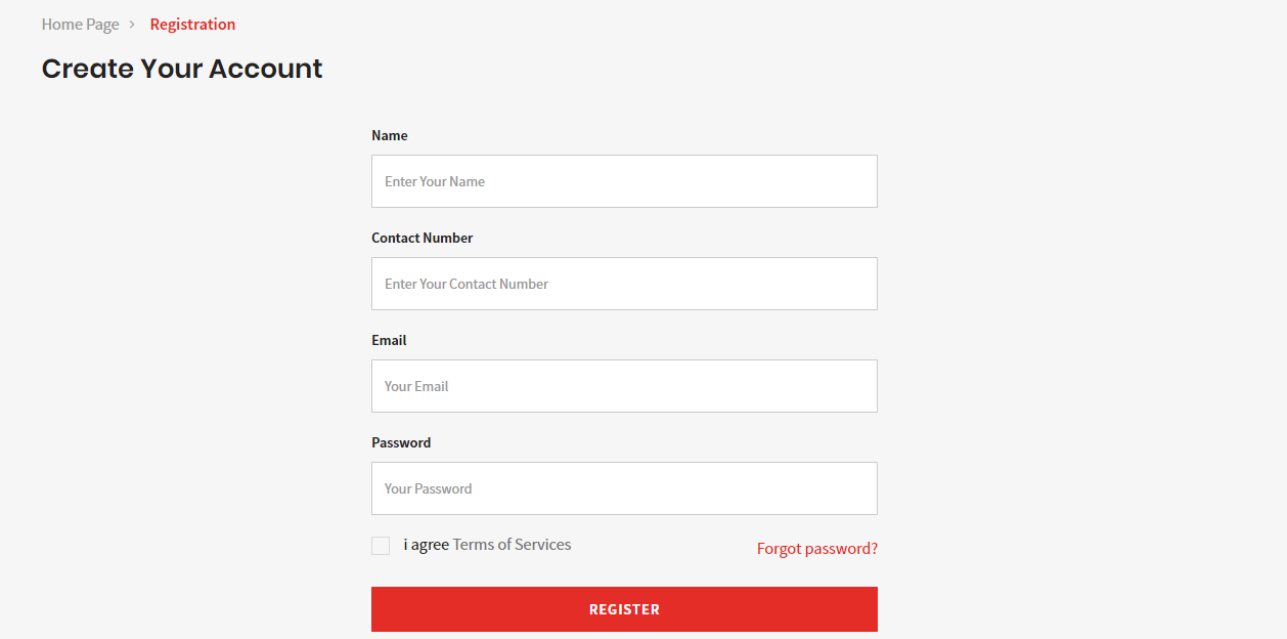

Рис.3.17. Сторінка реєстрації.

- Сторінка з формою для авторизації

Сторінка авторизації дає змогу користувачу ввійти в систему. Під час авторизації на бекенді створюється сессія, саме завдяки цьому користувач має доступ тільки до своїх данних. Сторінка авторизації зображено на рис. 3.15

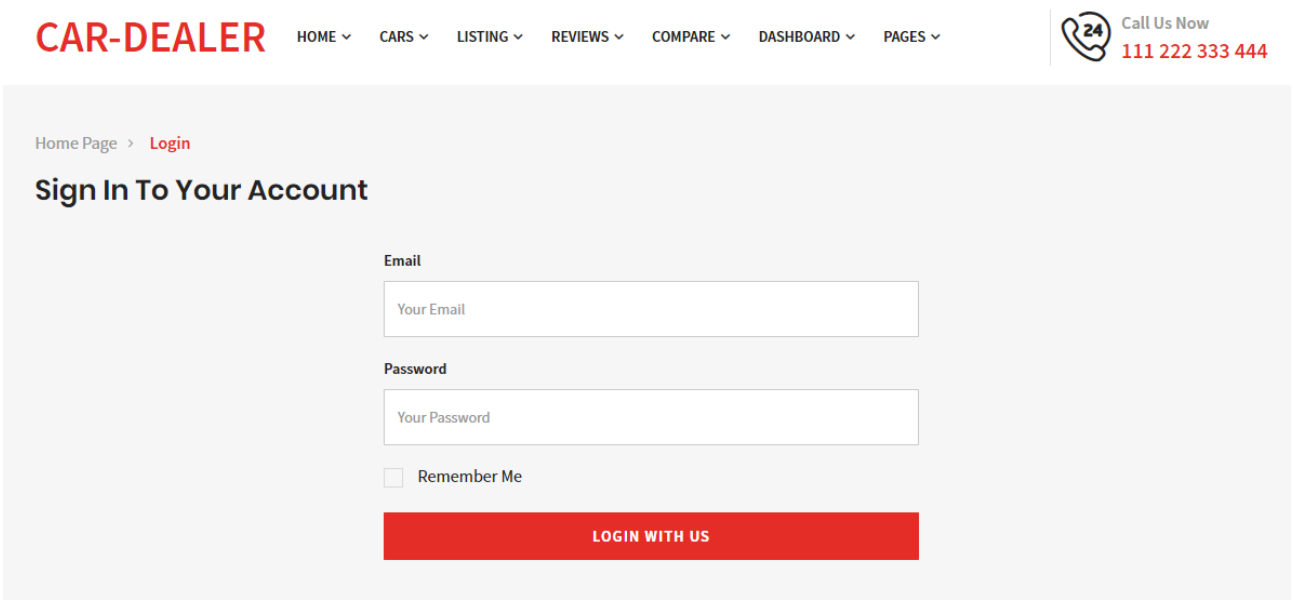

Рис. 3.18. Сторінка авторизації.

- Сторінка пошуку

Сторінка з пошуку (рис 3.16) дає можливість користувачу виконувати пошук автомобілей за ключовими словами.

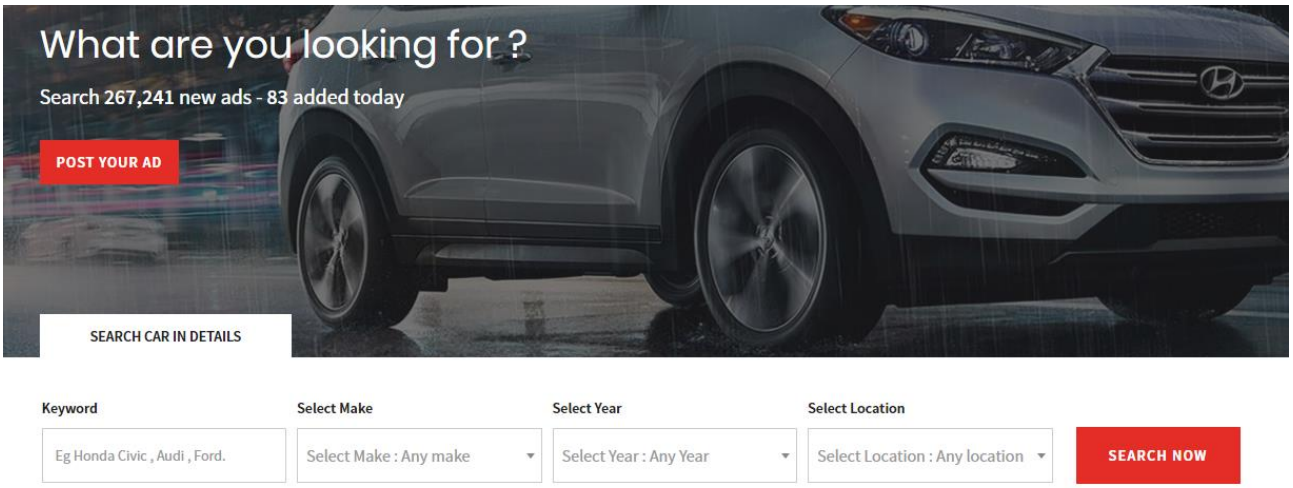

Рис. 3.19. Сторінка пошуку автомобіля

Результат виконання пошуку зображена на рис. 3.18 оголошення сортуються за датою створення та за актуальністю оголошення.

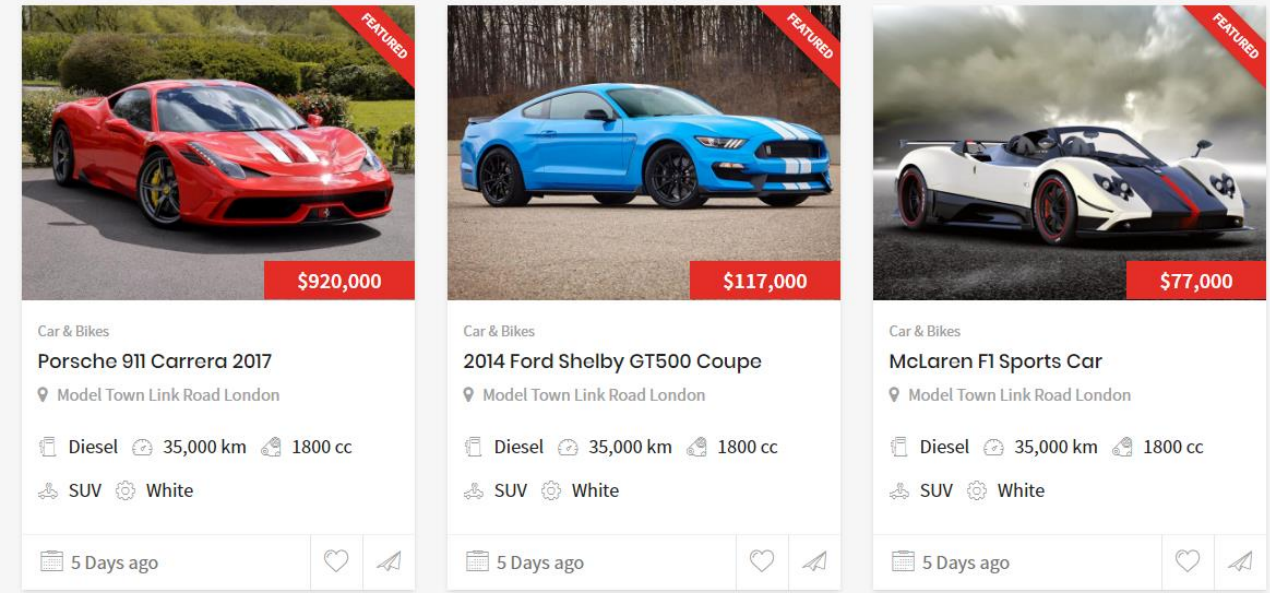

Рис. 3.20. Результат виконання пошуку

- Сторінка додавання оголошення

Сторінка додавання зображено на рис 3.19 тут користувач може редагувати свої оголошення, а також додавати нові автомобілі для продажу.

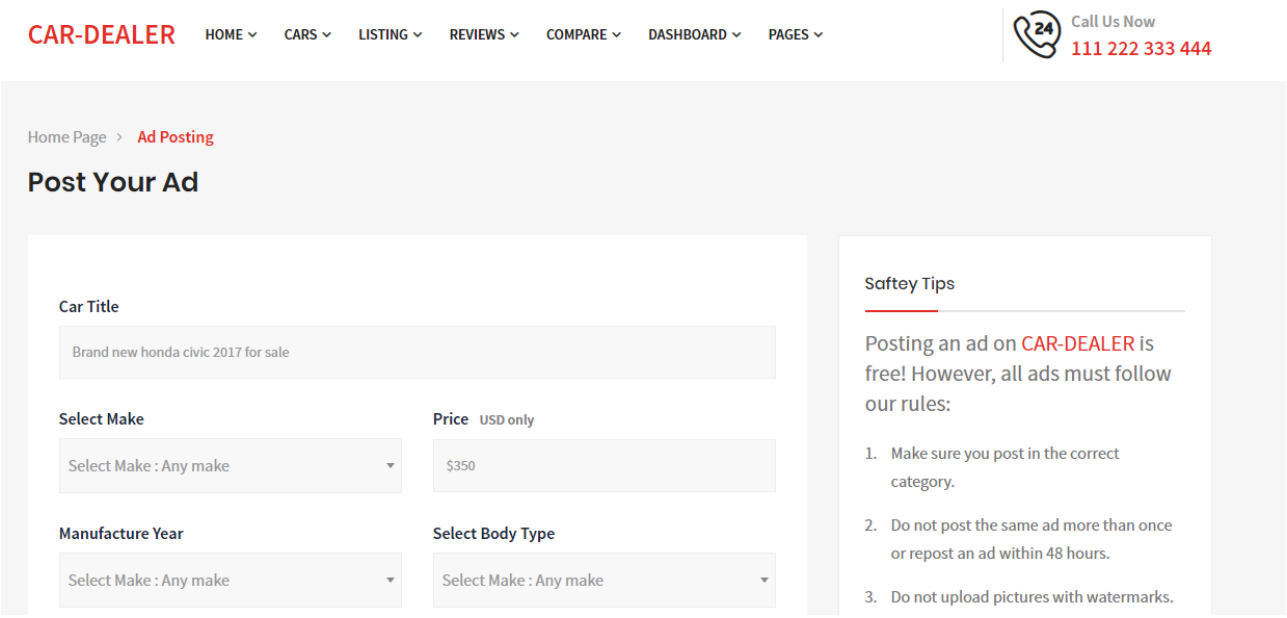

Рис. 3.21. Сторінка додавання нового оголошення

ВИСНОВОК ДО РОЗДІЛУ 3

Розроблений web-додаток car-dealer має зручний інтерфейс для користувача. Бекенд реалізує архітектуру MVC , всі сервісні класи слабо зв`язані, і це досить зручно для маштабування системи.

В результаті виконання проекту були виконані всі поставлені задачі. Всі сутності зберігаються до БД. Для web-додатоку були написані тести це дає змогу зрозуміти що система працює вірно.

ВИСНОВКИ

В ході виконання першої частини роботи – з обґрунтування розробки web-додатку – були розглянуті основні принци роботи web-додатка, основні ознаки та складові частини. Після проведення аналізу архітектури web-додатка, було виявлено, що саме паттерн MVC є найбільш гнучким та слабозв’язаною системою для web-додатку.

У другій частині дипломної роботи розглядаються особливості технології Spring Framework. Також був проведенний аналіз предметної області. Методологія Канбан є ефективним способом візуалізації процесу розробки проектів. Розглянуто вимоги до web-додатку Task list які відповідають сучасним вимогам та стандартам.

Завданням даного дипломного проекту було розробка бекенду web-додатка car-dealer, основна функція якого – зберігання та обробка інформації про хід процесу реалізації проекту. Для імплементації цього завдання були виконані наступні пункти:

1) аналіз та перегляд літературних джерел по темі дипломного проекту;

2) аналіз проектування архітектури web-додатка;

3) проектування логічної моделі бази даних;

4) проектування фізичної моделі бази даних;

5) створення БД;

6) створення web- додатку car-dealer.

7) налаштування середовища для web- додатку car-dealer.

Для проектування логічної та фізичної моделей бази даних використовувався CASE-засіб ERwinDataModele, який надає можливість автоматично згенерувати скрипт з усіма необхідними SQL-запитами для

створення реальної БД. Для створення додатку використовувалося інтегроване середовище розробки Intellij IDEA, мовою програмування була обрана Java.

Візуально додаток виглядає інтуїтивно-зрозуміло для будь-якого користувача, тому для його використання не потрібно проходити спеціальної підготовки. Користовач може шукати автомобілі як заключовими словами так і за детальним пошуком, сортувати найдені авто за ціною за роком випуску, за релевантністб оголошення, також може додавати редагувати свої оголошення, залишати коментарі під оголошеннями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Web-додаток [Інтернет-ресурс] / Режим доступу: https://uk.wikipedia.org/wiki/

2. Что такое веб-приложения и динамические веб-страницы [Інтернет-ресурс] / Режим доступу: https://helpx.adobe.com/ru/dreamweaver/using/web-applications.html

3. Model–view–controller [Інтернет-ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Model-view-controller

4. Spring Framework [Інтернет-ресурс] / Режим доступу: https://spring.io/

5. Kanban: как выполнять задачи точно в срок [Інтернет-ресурс] / Режимдоступу: http://www.alexcouncil.com/kanban/

6. Базы данных: модели, разработки, реализация / Т.С. Карпова. - СПб.: Питер, 2002. – 240 с.

ДОДАТОК А

```
CREATE DATABASE `car-dealer`;

USE `car-dealer`;

CREATE TABLE `role` (
  `role_id` int(11) NOT NULL AUTO_INCREMENT,
  `role` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`role_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `user` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `active` int(11) DEFAULT NULL,
  `login` varchar(255) NOT NULL UNIQUE,
  `password` varchar(255) NOT NULL,
  `first_name` varchar(255) NOT NULL,
  `second_name` varchar(255) NOT NULL,
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `user_role` (
  `user_id` int(11) NOT NULL,
  `role_id` int(11) NOT NULL,
  PRIMARY KEY (`user_id`,`role_id`),
  KEY `FKa68196081fvovjhkek5m97n3y` (`role_id`),
  CONSTRAINT `FK859n2jvi8ivhui0rl0esws6o` FOREIGN KEY (`user_id`) REFERENCES `user`
(`user_id`),
  CONSTRAINT `FKa68196081fvovjhkek5m97n3y` FOREIGN KEY (`role_id`) REFERENCES `role`
(`role_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `role` VALUES (1,'USER');

CREATE TABLE `advert`
(
    `advert_id` int(11) NOT NULL AUTO_INCREMENT,
    `user_id`   int(11) NOT NULL,
    `car_id`    int(11) NOT NULL,
    PRIMARY KEY (`advert_id`),
    FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`),
    FOREIGN KEY (`car_id`) REFERENCES `car` (`car_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `car`
(
    `car_id`      int(11)      NOT NULL AUTO_INCREMENT,
    `advert_id`   int(11)      NOT NULL,
    `model_id`    int(11)      NOT NULL,
    `currency_id` int(11)      NOT NULL,
    `year_id`     int(11)      NOT NULL,
    `price_id`    int(11)      NOT NULL,
    `description` varchar(255) NOT NULL,
    PRIMARY KEY (`car_id`),
    FOREIGN KEY (`advert_id`) REFERENCES `advert` (`advert_id`),
    FOREIGN KEY (`model_id`) REFERENCES `model` (`model_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `currency`
(
    `currency_id`   int(11)      NOT NULL AUTO_INCREMENT,
    `currency`      varchar(255) NOT NULL,
    `exchange_rate` decimal(255) NOT NULL,
    PRIMARY KEY (`currency_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `brand`
(
```

```
    `brand_id` int(11)       NOT NULL AUTO_INCREMENT,
    `brand`    varchar(255) NOT NULL,
    PRIMARY KEY (`brand_id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

CREATE TABLE `model`
(
    `model_id` int(11)       NOT NULL AUTO_INCREMENT,
    `brand_id` int(11)       NOT NULL,
    `model`    varchar(255) NOT NULL,
    PRIMARY KEY (`model_id`),
    FOREIGN KEY (`brand_id`) REFERENCES `brand` (`brand_id`)

) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

CREATE TABLE `photo`
(
    `photo_id`  int(11)       NOT NULL AUTO_INCREMENT,
    `advert_id` int(11)       NOT NULL,
    `path`      varchar(255) NOT NULL,
    `main`      boolean       NOT NULL,
    PRIMARY KEY (`photo_id`),
    FOREIGN KEY (`advert_id`) REFERENCES `advert` (`advert_id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

CREATE TABLE `comment`
(
    `comment_id` int(11)       NOT NULL AUTO_INCREMENT,
    `user_id`    int(11)       NOT NULL,
    `advert_id`  int(11)       NOT NULL,
    `comment`    varchar(255) NOT NULL,
    PRIMARY KEY (`comment_id`),
    FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`),
    FOREIGN KEY (`advert_id`) REFERENCES `advert` (`advert_id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

CREATE TABLE `contact`
(
    `contact_id` int(11)       NOT NULL AUTO_INCREMENT,
    `user_id`    int(11)       NOT NULL,
    `advert_id`  int(11)       NOT NULL,
    `type`       varchar(255) NOT NULL,
    `value`      varchar(255) NOT NULL,
    PRIMARY KEY (`contact_id`),
    FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;
```

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.nau.icit</groupId>
    <artifactId>task-list</artifactId>
    <version>1.0</version>
    <packaging>war</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.4.2.RELEASE</version>
        <relativePath/>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>6.0.8.Final</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>bootstrap</artifactId>
            <version>3.3.6</version>
        </dependency>
```

```
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-all</artifactId>
            <version>1.9.5</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <version>4.2.2.RELEASE</version>
            <scope>test</scope>
        </dependency>
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>7.0</version>
    <type>jar</type>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.21</version>
</dependency>
<dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
</dependency>
<dependency>
    <groupId>com.google.apis</groupId>
    <artifactId>google-api-services-calendar</artifactId>
    <version>v3-rev125-1.20.0</version>
    <exclusions>
        <exclusion>
            <groupId>com.google.guava</groupId>
            <artifactId>guava-jdk5</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>com.google.apis</groupId>
    <artifactId>google-api-services-drive</artifactId>
    <version>v3-rev110-1.23.0</version>
</dependency>

<dependency>
    <groupId>com.google.api-client</groupId>
    <artifactId>google-api-client</artifactId>
    <version>1.27.0</version>
</dependency>

<dependency>
    <groupId>com.google.oauth-client</groupId>
    <artifactId>google-oauth-client-jetty</artifactId>
    <version>1.23.0</version>
</dependency>

<dependency>
    <groupId>com.googlecode.owasp-java-html-sanitizer</groupId>
    <artifactId>owasp-java-html-sanitizer</artifactId>
    <version>20170515.1</version>
</dependency>
```

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.4</version>
</dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

ДОДАТОК В

```
spring.main.banner-mode=off
spring.mvc.view.prefix = /WEB-INF/views/
spring.mvc.view.suffix = .jsp

spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost/car-dealer
spring.datasource.username=root
spring.datasource.password=mysql

spring.queries.users-query=select login, password, active from user where
login=?
spring.queries.roles-query=select u.login, r.role from user u inner join
user_role ur on(u.user_id=ur.user_id) inner join role r on(ur.role_id=r.role_id)
where u.login=?
```

```
package com.nau.icit.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

import javax.sql.DataSource;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    private static final String LOGIN = "/login";

    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Autowired
    private DataSource dataSource;

    @Value("${spring.queries.users-query}")
    private String usersQuery;

    @Value("${spring.queries.roles-query}")
    private String rolesQuery;

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
            throws Exception {
        auth
                .jdbcAuthentication()
                .usersByUsernameQuery(usersQuery)
                .authoritiesByUsernameQuery(rolesQuery)
                .dataSource(dataSource)
                .passwordEncoder(bCryptPasswordEncoder);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                    .antMatchers("/webjars/**").permitAll()
                    .antMatchers("/").permitAll()
                    .antMatchers(LOGIN).permitAll()
                    .antMatchers("/registration").permitAll()
                    .anyRequest()
                    .authenticated()
                    .and()
                .csrf().disable()
```

```
                .formLogin()
                .loginPage(LOGIN)
                .defaultSuccessUrl("/board-list")
                .usernameParameter("login")
                .passwordParameter("password")
                .and()
            .logout()
                .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                .logoutSuccessUrl(LOGIN);
    }
}
```

Сутність Оголошення:

```
package com.nau.icit.model;

import javax.persistence.*;

@Entity
@Table(name = "advert")
public class Advert {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "advert_id")
    private Long id;

    @OneToOne
    @JoinColumn(name = "advert_id")
    private User user;

    @OneToOne
    @JoinColumn(name = "car_id")
    private Car car;

    public Advert() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public Car getCar() {
        return car;
    }

    public void setCar(Car car) {
        this.car = car;
    }
}
```

Сутність Користувач:

```
package com.nau.icit.model;

import javax.persistence.*;
import javax.validation.constraints.Email;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;
import java.util.Set;

@Entity
@Table(name = "user")
public class User {
```

```
    private static final String NAME_MESSAGE = "*Your name must have at least 3
characters";
    private static final String LOGIN_MESSAGE = "*Your login must have at least 4
characters";
    private static final String LOGIN_PATTERN = "^(?=.*[A-Za-z0-9]$)[A-Za-z][A-Za-z\\d.-
]{0,19}$";
    private static final String LOGIN_PATTERN_MESSAGE = "*You can use letters, numbers
and full stops";
    private static final String EMAIL_MESSAGE = "*Please provide your email in right
form.";
    private static final String ORDER_WORD_MESSAGE = "*Your password must have at least 5
characters";

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "user_id")
    private Long id;

    @Column(name = "login")
    @Size(min = 4, message = LOGIN_MESSAGE)
    @Pattern(regexp = LOGIN_PATTERN, message = LOGIN_PATTERN_MESSAGE)
    private String login;

    @Column(name = "email")
    @Email(message = EMAIL_MESSAGE)
    private String email;

    @Column(name = "password")
    @Size(min = 5, message = ORDER_WORD_MESSAGE)
    private String password;

    @Column(name = "first_name")
    @Size(min = 3, message = NAME_MESSAGE)
    private String firstName;

    @Column(name = "second_name")
    @Size(min = 3, message = NAME_MESSAGE)
    private String secondName;

    @Column(name = "active")
    private int active;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"),
inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
```

```
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getSecondName() {
        return secondName;
    }

    public void setSecondName(String secondName) {
        this.secondName = secondName;
    }

    public int getActive() {
        return active;
    }

    public void setActive(int active) {
        this.active = active;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }
}
```

Сутність Машина:

```
package com.nau.icit.model;

import javax.persistence.*;

@Entity
@Table(name = "car")
public class Car {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "car_id")
    private Long id;

    @OneToOne
    @JoinColumn(name = "advert_id")
    private Advert advert;

    @OneToOne
    @JoinColumn(name = "model_id")
    private Model model;

    @OneToOne
    @JoinColumn(name = "currency_id")
    private Currency currency;

    @Column(name = "price")
```

```
    private String price;

    @Column(name = "price")
    private String year;

    public Car() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Advert getAdvert() {
        return advert;
    }

    public void setAdvert(Advert advert) {
        this.advert = advert;
    }

    public Model getModel() {
        return model;
    }

    public void setModel(Model model) {
        this.model = model;
    }

    public Currency getCurrency() {
        return currency;
    }

    public void setCurrency(Currency currency) {
        this.currency = currency;
    }

    public String getPrice() {
        return price;
    }

    public void setPrice(String price) {
        this.price = price;
    }

    public String getYear() {
        return year;
    }

    public void setYear(String year) {
        this.year = year;
    }
}
```

Сутність Контакт:

```
package com.nau.icit.model;

import javax.persistence.*;

@Entity
@Table(name = "contact")
public class Contact {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "contact_id")
```

```
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @Column(name = "type")
    private String type;

    @Column(name = "value")
    private String value;

    public Contact() {
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

Сутність Коментарь:

```
package com.nau.icit.model;

import javax.persistence.*;

@Entity
@Table(name = "comment")
public class Comment {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "comment_id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "advert_id")
    private Advert advert;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
```

```
    @Column(name = "value")
    private String value;

    public Comment() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Advert getAdvert() {
        return advert;
    }

    public void setAdvert(Advert advert) {
        this.advert = advert;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```

Сутність Фотокартка:

```
package com.nau.icit.model;


import javax.persistence.*;

@Entity
@Table(name = "photo")
public class Photo {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "photo_id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "advert_id")
    private Advert advert;

    @Column(name = "path")
    private String path;

    @Column(name = "main")
    private Boolean main;

    public Photo() {
    }

    public Long getId() {
```

```
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Advert getAdvert() {
        return advert;
    }

    public void setAdvert(Advert advert) {
        this.advert = advert;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public Boolean getMain() {
        return main;
    }

    public void setMain(Boolean main) {
        this.main = main;
    }
}
```

```java
package com.nau.icit.repository;

import com.nau.icit.model.Advert;
import com.nau.icit.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface AdvertRepository extends JpaRepository<Advert, Long> {
    Advert findAdvertById(Long id);

    List<Advert> findAdvertsByUser(User user);
}
```

```java
package com.nau.icit.repository;
import com.nau.icit.model.Role;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RoleRepository extends JpaRepository<Role, Long> {
    Role findByRole(String role);
}
```

```java
package com.nau.icit.repository;

import com.nau.icit.model.Advert;
import com.nau.icit.model.Photo;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface PhotoRepository extends JpaRepository<Photo, Long> {
    Photo findTaskListById(Long id);

    List<Photo> findTaskListsByAdvert(Long id);

    void deleteAllByAdvert(Advert advert);
}
```

```java
package com.nau.icit.repository;

import com.nau.icit.model.Car;
import com.nau.icit.model.Task;
import com.nau.icit.model.TaskList;
import com.nau.icit.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CarRepository extends JpaRepository<Car, Long> {
    Car findCarById(Long id);

    void deleteAllByUser(User user);
}
package com.nau.icit.repository;
import com.nau.icit.model.Attachment;
import com.nau.icit.model.Task;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AttachmentRepository extends JpaRepository<Attachment, Long> {
    Attachment findAttachmentById(Long id);
    void deleteAllByTask(Task task);
}
```

```
package com.nau.icit.controller;

import com.nau.icit.model.User;
import com.nau.icit.service.UserAuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

import javax.validation.Valid;

@Controller
public class LoginController {

    private static final String REGISTRATION_SUCCESS = "Registration was
success, please login";
    private static final String LOGIN_ERROR =  "*There is already a user
registered with the login provided";
    private static final String LOGIN = "login";
    private static final String REGISTRATION = "registration";

    @Autowired
    private UserAuthService userAuthService;

    @GetMapping({"/", "/login"})
    public String login(){
        return LOGIN;
    }

    @GetMapping("/registration")
    public String registration(ModelMap modelMap){
        User user = new User();
        modelMap.addAttribute("user", user);
        return REGISTRATION;
    }

    @PostMapping("/registration")
    public String registration(@Valid User user, BindingResult bindingResult,
ModelMap modelMap) {
        User userExists = userAuthService.findUserByLogin(user.getLogin());
        if (userExists != null)
            bindingResult.rejectValue(LOGIN, "error.user", LOGIN_ERROR);

        if (bindingResult.hasErrors())
            return REGISTRATION;

        userAuthService.saveUser(user);
        modelMap.addAttribute("showMessage", true);
        modelMap.addAttribute("message", REGISTRATION_SUCCESS);
        return LOGIN;
    }

}
```

```
package com.nau.icit.controller;

import com.nau.icit.model.Advert;
import com.nau.icit.model.Comment;
import com.nau.icit.model.Photo;
import com.nau.icit.repository.AdvertRepository;
import com.nau.icit.repository.CommentRepository;
import com.nau.icit.repository.PhotoRepository;
import com.nau.icit.service.UserAuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

@Controller
public class AdvertController {

    @Autowired
    private UserAuthService userAuthService;
    @Autowired
    private AdvertRepository advertRepository;
    @Autowired
    private PhotoRepository photoRepository;
    @Autowired
    private CommentRepository commentRepository;

    @GetMapping("/advert")
    public String getAdbert(@RequestParam Long id, ModelMap modelMap) {
        List<Advert> list =
advertRepository.findAdvertsByUser(userAuthService.getAuthUser());
        modelMap.addAttribute("list", list);
        return "advert";
    }

    @Transactional
    @PostMapping("/advert")
    public String editAdvert(Advert advert, ModelMap modelMap) {
        advert.setUser(userAuthService.getAuthUser());
        advertRepository.save(advert);
        modelMap.clear();
        return "redirect:/board?id=" + advert.getId();
    }

    @Transactional
    @PostMapping("/add-photos")
    public String addPhotos(@RequestParam Long id, List<Photo> photos) {
        Advert advert = advertRepository.findAdvertById(id);
        photos.forEach(p -> p.setAdvert(advert));
        return "redirect:/advert?id=" + advert.getId();
    }


    @Transactional
    @GetMapping("/remove-advert")
    public String removeAdvert(@RequestParam Long id) {
        advertRepository.delete(id);
        return "redirect:/";
    }

    @Transactional
    @PostMapping("/add-comment")
    public String addComment(@RequestParam Long id, @RequestParam String value,
@RequestParam String login) {
        Comment comment = new Comment();
        comment.setValue(value);
        comment.setUser(userAuthService.findUserByLogin(login));
        comment.setAdvert(advertRepository.findOne(id));
```

```
        commentRepository.save(comment);
        return "redirect:/board?id=" + comment.getAdvert().getId();
    }
}
```

```
package com.nau.icit.controller;

import com.nau.icit.model.User;
import com.nau.icit.service.UserAuthService;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.web.servlet.View;

import static org.mockito.Matchers.anyString;
import static org.mockito.Mockito.when;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.model;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;
import static
org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSetup;

@RunWith(MockitoJUnitRunner.class)
public class LoginControllerTest {

    public static final String FIRST_NAME = "firstName";
    public static final String SECOND_NAME = "secondName";
    public static final String ORDER_WORD = "password";
    public static final String LOGIN = "login";
    public static final String EMAIL = "email";
    public static final String REGISTRATION = "registration";
    @InjectMocks
    private LoginController loginController;
    @Mock
    private View mockView;
    @Mock
    private UserAuthService userAuthService;
    private MockMvc mockMvc;
    private User invalidUser, validUser;

    @Before
    public void setUp(){
        mockMvc = standaloneSetup(loginController)
                .setSingleView(mockView)
                .build();
        invalidUser = new User();
        invalidUser.setId(1L);
        invalidUser.setFirstName("F");
        invalidUser.setSecondName("S");
        invalidUser.setLogin("L");
        invalidUser.setEmail("E");
        invalidUser.setPassword("P");

        validUser = new User();
        validUser.setId(2L);
```

```
        validUser.setFirstName("John");
        validUser.setSecondName("Doe");
        validUser.setLogin("john1990");
        validUser.setEmail("john1990@gmail.com");
        validUser.setPassword("john1990");
    }

    @Test
    public void shouldReturnToLoginPage() throws Exception {
        mockMvc.perform(get("/login"))
                .andExpect(status().isOk())
                .andExpect(view().name(LOGIN));
    }

    @Test
    public void shouldReturnToRegistrationPage() throws Exception {
        mockMvc.perform(get("/registration"))
                .andExpect(status().isOk())
                .andExpect(view().name(REGISTRATION));
    }

    @Test
    public void shouldReturnToRegistrationPageIfUserIsInvalid() throws Exception
{

when(userAuthService.findUserByLogin(invalidUser.getLogin())).thenReturn(invalid
User);
        mockMvc.perform(post("/registration")
                    .param(FIRST_NAME, invalidUser.getFirstName())
                    .param(SECOND_NAME, invalidUser.getSecondName())
                    .param(ORDER_WORD, invalidUser.getPassword())
                    .param(LOGIN, invalidUser.getLogin())
                    .param(EMAIL, invalidUser.getEmail()))
                .andExpect(model().errorCount(6))
                .andExpect(status().isOk())
                .andExpect(view().name(REGISTRATION));
    }

    @Test
    public void shouldReturnToLoginPageAfterSuccessfulRegistration() throws
Exception {
        when(userAuthService.findUserByLogin(anyString())).thenReturn(null);
        mockMvc.perform(post("/registration")
                    .param(FIRST_NAME, validUser.getFirstName())
                    .param(SECOND_NAME, validUser.getSecondName())
                    .param(ORDER_WORD, validUser.getPassword())
                    .param(LOGIN, validUser.getLogin())
                    .param(EMAIL, validUser.getEmail()))
                .andExpect(model().errorCount(0))
                .andExpect(model().hasNoErrors())
                .andExpect(status().isOk())
                .andExpect(view().name(LOGIN));
    }
}
```

```
package com.nau.icit.controller;

import com.nau.icit.model.*;
import com.nau.icit.repository.AdvertRepository;
import com.nau.icit.repository.PhotoRepository;
import com.nau.icit.repository.TaskRepository;
import com.nau.icit.service.UserAuthService;
```

```
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.ArgumentCaptor;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.web.servlet.View;

import java.util.ArrayList;
import java.util.List;

import static junit.framework.TestCase.assertEquals;
import static junit.framework.TestCase.assertNull;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;
import static org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSetup;

@RunWith(MockitoJUnitRunner.class)
public class AdvertControllerTest {

    @InjectMocks
    private AdvertController advertController;
    @Mock
    private View mockView;
    @Mock
    private AdvertRepository advertRepository;
    @Mock
    private PhotoRepository taskListRepository;
    @Mock
    private UserAuthService userAuthService;
    @Mock
    private TaskRepository taskRepository;
    private MockMvc mockMvc;
    private User user;
    private List<Advert> adverts;
    private Advert advert;
    private List<Photo> photos;
    private Task task;

    @Before
    public void setUp() {
        mockMvc = standaloneSetup(advertController)
                .setSingleView(mockView)
                .build();
        user = new User();
        Advert advert = new Advert();
        adverts = new ArrayList<>();
        adverts.add(advert);
    }

    @Test
    public void shouldReturnToAdvertsPage() throws Exception {
        when(advertRepository.findAll()).thenReturn(adverts);

        mockMvc.perform(get("/advert?id=" + advert.getId()))
                .andExpect(status().isOk())
                .andExpect(view().name("advert"));
    }

    @Test
    public void shouldEditAdvertNameAndReturnToAdvertPage() throws Exception{
        when(userAuthService.getAuthUser()).thenReturn(user);
        ArgumentCaptor<Advert> captor = ArgumentCaptor.forClass(Advert.class);

        mockMvc.perform(post("/advert")
                    .param("id", advert.getId().toString())
                    .param("name", "Fiat"))
```

```
                .andExpect(status().isOk())
                .andExpect(view().name("redirect:/advert?id=" + advert.getId()));

        verify(advertRepository).save(captor.capture());
        assertEquals(advert.getId(), captor.getValue().getId());
        assertEquals("Fiat",
captor.getValue().getCar().getModel().getBrand().getBrand());

    }

    @Test
    public void shouldAddPhotostAndReturnToAdvertPage() throws Exception{
        when(advertRepository.findAdvertsByUser(user).get(0)).thenReturn(advert);
        ArgumentCaptor<Advert> captor = ArgumentCaptor.forClass(Advert.class);

        mockMvc.perform(post("/add-photos?id=" + advert.getId()))
                .andExpect(status().isOk())
                .andExpect(view().name("redirect:/advert?id=" + advert.getId()));

        verify(advertRepository).save(captor.capture());
    }

}
```

```
package com.nau.icit.controller;

import com.nau.icit.model.Task;
import com.nau.icit.repository.TaskRepository;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.web.servlet.View;

import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSetup;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;

@RunWith(MockitoJUnitRunner.class)
public class TaskControllerTest {
    @InjectMocks
    private TaskController taskController;
    @Mock
    private View mockView;
    @Mock
    private TaskRepository taskRepository;
    private MockMvc mockMvc;
    private Task task;


    @Before
    public void setUp() {
        mockMvc = standaloneSetup(taskController)
                .setSingleView(mockView)
                .build();
        task = new Task();
        task.setId(1L);
        task.setPriority(1);
        task.setName("Spring Boot.");
        task.setDescription("Spring Boot is Spring's convention-over-configuration
solution for creating stand-alone.");
    }

    @Test
    public void shouldReturnToTaskPage() throws Exception {
        when(taskRepository.findTaskById(task.getId())).thenReturn(task);
        mockMvc.perform(get("/task?id=" + task.getId()))
```

```
                .andExpect(status().isOk())
                .andExpect(view().name("task"));
    }
}
package com.nau.icit.controller;

import com.nau.icit.model.Board;
import com.nau.icit.model.Task;
import com.nau.icit.model.TaskList;
import com.nau.icit.model.User;
import com.nau.icit.repository.AdvertRepository;
import com.nau.icit.repository.PhotoRepository;
import com.nau.icit.repository.TaskRepository;
import com.nau.icit.service.UserAuthService;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.ArgumentCaptor;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.web.servlet.View;

import java.util.ArrayList;
import java.util.List;

import static junit.framework.TestCase.assertEquals;
import static org.hamcrest.Matchers.hasSize;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.model;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;
import static org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSetup;

@RunWith(MockitoJUnitRunner.class)
public class BoardListControllerTest {

    @InjectMocks
    private BoardListController boardListController;
    @Mock
    private View mockView;
    @Mock
    private AdvertRepository boardRepository;
    @Mock
    private PhotoRepository taskListRepository;
    @Mock
    private UserAuthService userAuthService;
    @Mock
    private TaskRepository taskRepository;
    private MockMvc mockMvc;
    private User user;
    private List<Board> boards;
    private TaskList taskList;
    private List<TaskList> lists;
    private Board board;
    private Task task;

    @Before
    public void setUp() {
        mockMvc = standaloneSetup(boardListController)
                .setSingleView(mockView)
                .build();
        user = new User();
        board = new Board();
        board.setId(1L);
        board.setUser(user);
        board.setName("Java");
        boards = new ArrayList<>();
```

```
        boards.add(board);
        boards.add(new Board());
        taskList = new TaskList();
        taskList.setId(4L);
        taskList.setName("To do");
        taskList.setBoard(board);
        lists = new ArrayList<>();
        lists.add(taskList);
        task = new Task();
        task.setId(5L);
        task.setName("JPA");
        task.setTaskList(taskList);
    }

    @Test
    public void shouldReturnToBoardsPage() throws Exception{
        when(userAuthService.getAuthUser()).thenReturn(user);
        when(boardRepository.findBoardsByUser(user)).thenReturn(boards);
        mockMvc.perform(get("/board-list"))
                .andExpect(status().isOk())
                .andExpect(model().attribute("boards", hasSize(2)))
                .andExpect(view().name("board-list"));
    }

    @Test
    public void shouldSaveBoardAndReturnToBoardsPage() throws Exception{
        when(userAuthService.getAuthUser()).thenReturn(user);
        ArgumentCaptor<Board> captor = ArgumentCaptor.forClass(Board.class);

        mockMvc.perform(post("/board-list")
                    .param("name", "Java"))
                .andExpect(status().isOk())
                .andExpect(view().name("redirect:/board-list"));

        verify(boardRepository).save(captor.capture());
        assertEquals(board.getName(), captor.getValue().getName());
    }

    @Test
    public void shouldRemoveBoardAndReturnToBoardsPage() throws Exception{
        when(taskListRepository.findTaskListsByBoardId(board.getId())).thenReturn(lists);
        when(boardRepository.findBoardById(board.getId())).thenReturn(board);

        mockMvc.perform(get("/remove-board?id=" + board.getId()))
                .andExpect(status().isOk())
                .andExpect(view().name("redirect:/board-list"));

        verify(taskRepository).deleteAllByTaskList(taskList);
        verify(taskListRepository).deleteAllByBoard(board);
        verify(boardRepository).delete(board.getId());
    }
}
@Service
public class FileService {

    @Autowired
    private FileLinkRepo fileLinkRepo;
    @Autowired
    private AppStorageClient fileStorage;
    @Autowired
    protected SolrService solrService;

    public static final Cache<String, List<String>> fileContentCache
            = CacheBuilder.newBuilder().expireAfterWrite(10,
TimeUnit.MINUTES).maximumSize(100).build();

    private static final String source = "CleverStaff";
    private static final String type = "CleverStaff";
    private static final HttpConnector HTTP_CONNECTOR = new HttpConnector();
    private static final Set<String> textFormatExtension = new
HashSet<>(Arrays.asList("pdf", "doc", "docx", "txt", "rtf",
            "odt", "htm", "html"));
```

```
    public String getLinkById(final String atmId, final String session) {
        return fileStorage.getFileUrl(atmId, session);
    }

    public byte[] getFileInByteFromStorage(String requestLink) throws IOException {
        URLConnection connection;
        connection = HTTP_CONNECTOR.getConnection(requestLink, "GET");
        connection.setRequestProperty("Content-Type", "application/octet-stream");
        return IOUtils.toByteArray(connection.getInputStream());
    }

    public byte[] getFileBytesFromStorage(String attId) throws IOException {
        return IOUtils.toByteArray(getInputStreamFromStorage(attId));
    }

    public InputStream getInputStreamFromStorage(String attId) throws IOException {
        String url = getLinkByIdDirectlyWithoutSession(attId);
        URLConnection connection;
        connection = HTTP_CONNECTOR.getConnection(url, "GET");
        connection.setRequestProperty("Content-Type", "application/octet-stream");
        return connection.getInputStream();
    }

    public Pack getListFiles(Collection<String> attIds) throws IOException {
        return fileStorage.getList(new ArrayList<>(attIds));
    }

    public String getLinkByIdDirectlyWithoutSession(String attmId) {
        return fileStorage.getFileUrl(attmId, null);
    }

    public String getLinkByIdandSession(String atmId) {
        return fileStorage.getFileUrl(atmId, SecurityUtil.getSessionId());
    }

    public void remove(FileLink fileLink) {
        fileLinkRepo.delete(fileLink.getFileId());
        if (fileLink.getType().equalsIgnoreCase(AttachTypeEnum.file.toString())) {
            try {
                fileStorage.removeAttachment(fileLink.getFileId(),
SecurityUtil.getUserId());
            } catch (Exception e) {
            }
        }
    }

    public void remove(List<FileLink> fileLinks) {
        new Thread(() -> {
            fileLinkRepo.delete(fileLinks);
            for (FileLink fileLink : fileLinks) {
                if (fileLink.getType().equalsIgnoreCase(AttachTypeEnum.file.toString()))
{
                    try {
                        fileStorage.removeAttachment(fileLink.getFileId(), "public");
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }).start();
    }

    public void removeFromAS(final String atmId) {
        try {
            fileStorage.removeAttachment(atmId, "public");
        } catch (Exception e) {
        }
    }

    public void removeFromASByLogin(final String atmId, final String login) {
        try {
```

```
            fileStorage.removeAttachment(atmId, login);
        } catch (Exception e) {
        }
    }

public void removeFromAS(final String atmId) {
    try {
        fileStorage.removeAttachment(atmId, "public");
    } catch (Exception e) {
    }
}

public void removeFromASByLogin(final String atmId, final String login) {
    try {
        fileStorage.removeAttachment(atmId, login);
    } catch (Exception e) {
    }
}

public boolean isTextFileByExtension(MultipartFile file) {
    return isTextFileByExtension(getFileName(file));
}

public boolean isTextFileByExtension(String fileName) {
    return textFormatExtension.contains(FileUtil.getExtensionByFileName(fileName));
}

public FileLink createFileLink(MultipartFile file, final ObjectEnum objName, final String
objId, final String userId, final String orgId) {
    if(file.getSize() == 0 ){
        throw new InfoException(ExceptionTypeEnum.textInFileIsEmpty);
    }
    List<FileLink> existsFiles = getFiles(objName, objId);
    List<String> existsFileIds = existsFiles.stream()
            .map(fl -> fl.getFileId()).collect(Collectors.toList());
    String saveId = saveFileInAppstorage(file, userId, objId,
            AccessTypeEnum.permitAll, existsFileIds);
    if (existsFileIds.contains(saveId)) {
        throw new InfoException(ExceptionTypeEnum.existsFile);
    }
    FileLink fl = new FileLink(saveId);
    fl.setCreatorId(userId);
    fl.setDc(new Date());
    fl.setFileName(getFileName(file));
    String fileExtension = FileUtil.getExtensionByFileName(getFileName(file));
    fl.setMime(MimeTypes.getMimeType(fileExtension));
    fl.setObjId(objId);
    fl.setObjName(objName.toString());
    fl.setOrgId(orgId);
    fl.setType(AttachTypeEnum.file.toString());
    return fileLinkRepo.save(fl);
}

public void saveListFileLinks(List<Attachment> attachmentsToSave, final ObjectEnum
objName, final String userId, final String orgId) {
    Pack pack = saveListFiles(attachmentsToSave, userId);
    List<FileLink> links = new ArrayList<>();
    for (Attachment attach : pack.attachments) {
        FileLink fl = new FileLink(attach.atmId);
        fl.setCreatorId(userId);
        fl.setDc(new Date());
        fl.setFileName(attach.fileName);
        String fileExtension = FileUtil.getExtensionByFileName(attach.fileName);
        fl.setMime(MimeTypes.getMimeType(fileExtension));
        fl.setObjId(attach.extId);
        fl.setObjName(objName.toString());
        fl.setOrgId(orgId);
        fl.setType(AttachTypeEnum.file.toString());
        links.add(fl);
    }
    fileLinkRepo.save(links);
}
```

```
public FileLink createFileLinkOrReturnExists(MultipartFile file, final ObjectEnum
objName, final String objId,
                                              final String userId, final String orgId) {
    String fileExtension = FileUtil.getExtensionByFileName(getFileName(file));
    String mimeType = MimeTypes.getMimeType(fileExtension);
    List<FileLink> existsFiles = getFiles(objName, objId);
    List<String> existsFileIds = existsFiles.stream()
            .map(fl -> fl.getFileId()).collect(Collectors.toList());
    String saveId = saveFileInAppstorage(file, userId, objId,
            AccessTypeEnum.permitAll, existsFileIds);
    if (existsFileIds.contains(saveId)) {
        existsFiles.stream().filter(fileLink ->
fileLink.getFileId().equalsIgnoreCase(saveId))
                .findFirst().orElseThrow(() -> new
InfoException(ExceptionTypeEnum.existsFile));
    }
    FileLink fl = new FileLink(saveId);
    fl.setCreatorId(userId);
    fl.setDc(new Date());
    fl.setFileName(getFileName(file));
    fl.setMime(mimeType);
    fl.setObjId(objId);
    fl.setObjName(objName.toString());
    fl.setOrgId(orgId);
    fl.setType(AttachTypeEnum.file.toString());
    FileLink save = fileLinkRepo.save(fl);
    return save;
}

public FileLink createFileLink(String atmId, String fileName, final ObjectEnum objName,
final String objId, String orgId, String personId) {
    String originalFileExtension = "";
    if (fileName.lastIndexOf(".") != -1) {
        originalFileExtension = fileName.substring(fileName.lastIndexOf(".") + 1);
        originalFileExtension = originalFileExtension.toLowerCase();
    }
    String mimeType = MimeTypes.getMimeType(originalFileExtension);
    List<FileLink> existsFiles = getFiles(objName, objId);
    List<String> existsFileIds = existsFiles.stream()
            .map(fl -> fl.getFileId()).collect(Collectors.toList());
    Attachment attachment = new Attachment();
    attachment.atmId = atmId;
    attachment.existAtmIds = existsFileIds;
    String saveId = fileStorage.getExistsAttachmentIdForIdsAndAtmId(attachment);
    if (existsFileIds.contains(saveId)) {
        return existsFiles.stream().filter(fl -> fl.getFileId().equalsIgnoreCase(saveId))
                .findFirst().get();
    }
    FileLink fl = new FileLink(atmId);
    fl.setCreatorId(personId);
    fl.setDc(new Date());
    fl.setFileName(fileName);
    fl.setMime(mimeType);
    fl.setObjId(objId);
    fl.setObjName(objName.toString());
    fl.setOrgId(orgId);
    fl.setType(AttachTypeEnum.file.toString());
    FileLink save = fileLinkRepo.save(fl);
    return save;
}
```